Representing Information using Parametric Visual Effects on Groupware Avatars

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Shane Dielschneider

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Parametric visual effects such as texture generation and shape grammars can be controlled to produce visually perceptible variation. This variation can be rendered on avatars in groupware systems in real time to represent user information in online environments. This type of extra information has been shown to enrich recognition and characterization, but has previously been limited to iconic representations. Modern, highly graphical virtual worlds require more naturalistic and stylistically consistent techniques to represent information.

A number of different parametric texture generation techniques are considered and a set of texture characteristics are developed. The variations of these texture characteristics are examined in a study to determine how well users can recognize the visual changes in each. Another study is done to determine how much screen space is required for users to recognize these visual changes in a subset of these texture characteristics.

Additionally, an example shape generation system is developed as an example of how shape grammars and L-systems can be used to represent information using a space ship metaphor.

These different parametric visual effects are implemented in an example prototype system using space ships. This prototype is a complete functioning groupware application developed in XNA that utilizes many parametric texture and shape effects.

Acknowledgements

I would like to thank my supervisors Dr. Carl Gutwin and Dr. David Mould whose knowledge, help, and patience made this thesis possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Real time distributed groupware allows interaction in virtual shared spaces, and visual representations of people can be used in these systems to provide a virtual "embodiment" or visual representation of each user. At a minimum these embodiments typically convey presence, identity, current activity, and movement. However, they are often limited in their ability to represent additional information, especially when compared to the amount of contextual information available in typical real-world, face-to-face interactions. This additional contextual information can have many uses, ranging from supporting physical task coordination to providing important social cues. Virtual embodiments often have very little support for this type of communication. One solution to this problem is providing "rich embodiments" which expand the use of embodiments to include additional information-rich components.

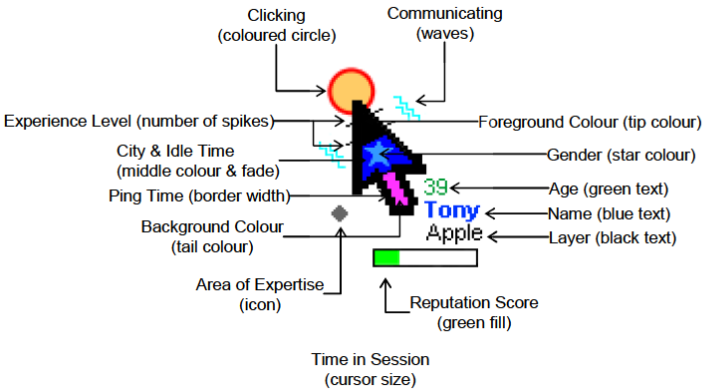


Figure 1.1: A remote cursor or "telepointer" rich embodiment. [36]

within Rich embodiments utilize glyphs, shapes, text, bar graphs, or other information visualization techniques to encode information within the visual represen-

tation of the embodiment itself. This additional information has been shown to be useful to enhance characterization and recognition as well as enriching interaction within virtual shared spaces [36]. However, many techniques that allow additional information to be presented on rich embodiments create graphical clutter, which increases user distraction and confusion. Additionally, these rich embodiments that rely on symbolic techniques such as text, bar graphs, and simple shapes can create embodiments that would be stylistically inconsistent with more naturalistic virtual worlds (see Figure 1.1).

A visual embodiment can be a very personal statement about the user, especially in highly social systems such as massively multi-player online role-playing games (MMORPGs) or virtual worlds such as Second Life (www.secondlife.com), and the embodiment's visual appeal is an important aspect of its function within the system. The visual appearance of an embodiment is vital to a user's virtual identity and his or her immersion within a highly graphical virtual world. Rich embodiments that rely upon iconic graphical methods are not compatible with the types of immersive and engaging embodiments that many users would prefer. Graphically simplistic methods have limited use in situations where graphically sophisticated embodiments are required and so the benefits of additional information cannot be realized in these systems.

1.1 Research Problem

The problem addressed in this thesis is that of: *Finding stylistically consistent techniques that can be used in highly naturalistic and immersive virtual worlds.*

Existing visual representations of information on groupware embodiments can be cluttered, distracting, and graphically simplistic. They utilize icons, text, or bars which obscure the environment and make stylistically inconsistent embodiments within more naturalistic virtual worlds. These embodiments have been shown to functionally enhance interaction within virtual spaces, but their graphical shortcomings limit their application in many systems which require highly naturalistic

or graphically developed embodiments. It is difficult to maximize the number of variables that are clearly represented on an embodiment while maintaining the level of graphical sophistication required for an embodiment in a graphically engaging virtual world.

1.2 Motivation

Providing additional information can improve recognition and characterization of users in a groupware system and facilitate or enrich interactions. In collaborative systems, information such as the expertise of a user could let other users know who to ask for help with a specific task. Knowledge of a user's current activity aids in coordination and avoids harmful interruptions. In multi-player gaming systems this additional information can inform player tactics: for example, more powerful enemies should be avoided or may require that players with complementary abilities team up and defeat them. Distinct embodiments allow fast and easy recognition of other players' avatars to either reciprocate past help or exact revenge for past misdeeds. Regardless of the application, rich embodiments provide additional information which informs people's actions within a shared virtual space.

Methods used to accomplish this information representation must be highly naturalistic and stylistically consistent to be applied in many popular systems which feature graphically complex worlds. The ubiquity of graphically powerful computers has led to widespread and massive popularity of applications and games featuring highly realistic, immersive three-dimensional environments. These compelling virtual worlds require equally compelling embodiments to maintain the high level of immersion. Simple graphical effects such as bar graphs, text, or simple icons conflict with the graphical metaphors of these systems. They can only be used sparingly and are often reserved for essential vital information such as name, status, speech bubbles, or other information of central importance to the system's purpose. To expand the number and types of information represented in these systems, more sophisticated graphical techniques are required. Additionally, the personal investment of a

user in a game character or alter ego may be substantial and his or her embodiment is a vital aspect of that identity. The visual and graphical appearance of the embodiment becomes a vital component of the system and any methods to introduce more information-rich embodiments cannot compromise the ability to generate visually consistent embodiments.

1.3 Solution

The solution to the problem of excessively cluttered and inconsistent embodiments is to find better ways to represent information through parameterized visual effects, focusing on two techniques: parametric texture generation and procedural shape generation. Parametric texture generation techniques allow realistic and naturalistic textures to be generated; the visually important parameters that control these techniques can be exploited as a medium to convey information. Shapes can be generated procedurally using L-systems or generative shape grammars which have been widely used to generate naturalistic structures including plants, trees, and other fractal patterns. The amount of growth and the types of shape rules applied can be controlled by parameters that encode information within the shape of an embodiment.

1.4 Steps in the Solution

The steps to the solution in this thesis include the following:

1. Develop a set of texture exemplars, each representing a controllable and perceptual texture characteristic. To represent information with texture it is first necessary to find expressive and distinctive textures that can be used. The result of this step is a set of texture characteristics which provide a useful base to develop future rich embodiments.

2. Develop a method of procedurally generating a shape that encodes a set of parameters. A method of shape generation must have the capacity to represent one or more variables to be useful in creating rich embodiments. The result of

this step is an example of a shape generation method that encodes a number of parameters in a naturalistic and visually meaningful way.

3. Determine perceptual limits of textures. The set of texture characteristics is empirically tested with users to determine the potential information capacity of each characteristic. It is necessary to establish the relationship between the generation parameters and the perceptible change within the texture so appropriate values can be represented. This step results in a mapping between perceived value and input parameter that can be used to inform development of rich embodiments.

4. Implement rich embodiments using textured representations and shape generation into a groupware system. A groupware system is implemented that uses the previously developed techniques. This system provides a proof-of-concept as well as an example for other systems to follow. It can also serve as a starting point for future work in this area.

5. Create guidelines for creating rich embodiments. These guidelines are informed by the development of the texture and shape techniques and through the lessons learned in the implementation of the groupware system. The result of this step is a set of advice and recommendations for designers using texture and shape to develop rich embodiments.

1.5 Contributions

The primary contribution of this thesis is a set of guidelines for development and improvement of rich embodiment using parametric visual effects. Other contributions include:

- An exploratory texture library
- Texture characteristics with exemplars
- Example parametric shape generation of avatars

- A study examining the relationship between texture parameters and user perception
- Example implementation of rich embodiments within a groupware system

1.6 Thesis Outline

The thesis is arranged as follows:

Chapter 2 includes a literature review that covers the following areas: groupware, information visualization, parametric texture generation, and procedural shape generation.

Chapter 3 includes information on the development and selection of texture exemplars. This includes the selection of texture generation methods, an exploratory texture library implementation, and the algorithm involved in generating the set of texture exemplars. This chapter also includes an implementation of a shape grammar that generates parametric outlines for generic spaceships.

Chapter 4 covers the methodology, design, and results of two studies to provide insight into using texture to represent information. The first study covers all the texture exemplars and the second looks specifically at the effect of texture swatch size.

Chapter 5 describes an exploratory prototype embodiment system as well as a functioning dog-fighting groupware system developed in XNA. The chapter also contains examples of rich embodiments generalized to different types of situations and domains such as biofeedback in a poker game and an internet discussion forum.

Chapter 6 includes the discussion and the development of design guidelines for rich embodiments.

Chapter 7 summarizes the thesis and its contributions as well as providing insight into possible future work.

Chapter 2

Review of Literature

The development of information-rich textured embodiment relies upon several areas of research: Groupware, Information Visualization, Parametric Texture Generation, and Procedural Shape Generation.

2.1 Groupware

Groupware is any computer-based system that supports a group of people engaged in a common task and provides an interface to a shared environment [13]. Ellis et al. distinguish between types of groupware according to their immediacy of interaction and distance between participants [13] (see Table 2.1). A distinction is made between "synchronous" and "asynchronous" groupware applications. Synchronous or "real-time" applications facilitate interactions that occur at the same time for all participants. Examples of synchronous applications are chat programs or multiplayer games. Asynchronous applications facilitate interactions that occur over a longer period of time with users accessing the system at different times. Forums and blogs are commonly used asynchronous groupware applications.

	Same Time	Different Times
Same Place	Face-to-face interaction.	Asynchronous interaction.
Different Place	Synchronous distributed interaction.	Asynchronous distributed interaction.

Table 2.1: Taxonomy of Groupware Interaction

The second way groupware applications are categorized is through the physical distance of the users and these are grouped into co-located and distributed groupware. Co-located groupware is used when all the users are in the same physical location. An example of co-located groupware is a system designed for a single shared display [37]. Distributed groupware supports a shared virtual environment for users in different locations across distances. An example of distributed groupware is a chat program in a multi-player game.

This thesis is concerned with real-time distributed groupware where there is fast-paced interaction that would benefit from visual embodiments that convey state information about users quickly and easily.

2.1.1 Embodiment

		Placement	
		Situated	Separate
Presentation	Literal		
	Symbolic		

Table 2.2: Presentation and Placement of Awareness Display Techniques

Real-time groupware systems often use a visual representation or embodiment to provide virtual personification of the user. Embodiments also have the ability to provide other information about the user such as presence, location, identity, and activity [3].

An organization for the display of embodiments was proposed by Gutwin and Greenberg [17] which categorizes them based upon two dimensions: placement and presentation (see Table 2.2). Presentation is broken into literal and abstract. A literal presentation represents the information as it is obtained, such as a video feed of a remote user [39]. Conversely, an abstract representation will present the information in a more symbolic way, such as using an hourglass icon to indicate a user is busy. The placement dimension is broken into situated and separate. A

situated embodiment will appear within the workspace such as a humanoid avatar within a multi-player game. An embodiment that is separate will appear somewhere outside the workspace such as on a radar view or mini map.

Avatars

Figure 2.1: Avatars from Second Life illustrating the variety of customization options available (www.secondlife.com).

This research is concerned with a literal and situated embodiment type commonly known as an avatar [28]. An avatar is typically a humanoid representation of a user within a virtual world although it can also include other mechanical, animal, or vehicular representations such as cars or spacecraft [36]. Within virtual worlds an avatar is chosen to represent a desired identity and its appearance is important for user status, self-disclosure, confidence, and socialization [28].

Customization

Modern video games have widely explored avatar customization for personalization purposes. The ability to customize an avatar to reflect a physical or mental ideal is important to users' overall satisfaction [11]. Users value the ability to modify avatar features that are most readily visible and commonly modified in real life, such as hair [11].

For example, Second Life is an online social virtual world that provides a large amount of customization to users (see Figure 2.1). Users can choose many types of

Figure 2.2: Avatars from Spore. Clockwise from top-left: dangerous flying creature, alert creature, fast creature with wings, creature with articulated hands (www.spore.com).

clothing and hairstyles as well as being able to incorporate animal, science fiction, or fantasy elements into their avatars.

A recent example of a game that has implemented highly customizable avatars that are designed to reflect game variables is Spore (www.spore.com). A primary feature of Spore is developing and customizing a race of creatures through a player-guided "evolution". The avatars do not change to reflect the statistics of the creatures, but the statistics of the creature change to reflect the players' selection of component body parts appearance. The appearance of a creature can indicate its rating in a few statistics such as speed, offense, and charm as well as the presence or absence of certain abilities such as jumping, flight, or even sight (see Figure 2.2).

Figure 2.3: Character avatars from Fable II. Clockwise from top left: character with blue magic runes, brawny and large character, an evil character, a good character. (www.lionhead.com)

Representing Information

Certain features of an avatar's appearance are sometimes controlled by the software to represent user or character information.

A modern video game that provides dynamic avatars is Fable II by Lionhead studios (www.Lionhead.com). Fable II avatars are designed to reflect the character's game stats: strong characters are larger while magic-using characters develop glowing blue runes on their skin. Evil characters have pale skin, black hair, and red eyes while good characters have tanned skin, light hair, blue eyes, and sometimes sport halos (see Figure 2.3). Characters also get old as time passes and gain weight based on their food choices.

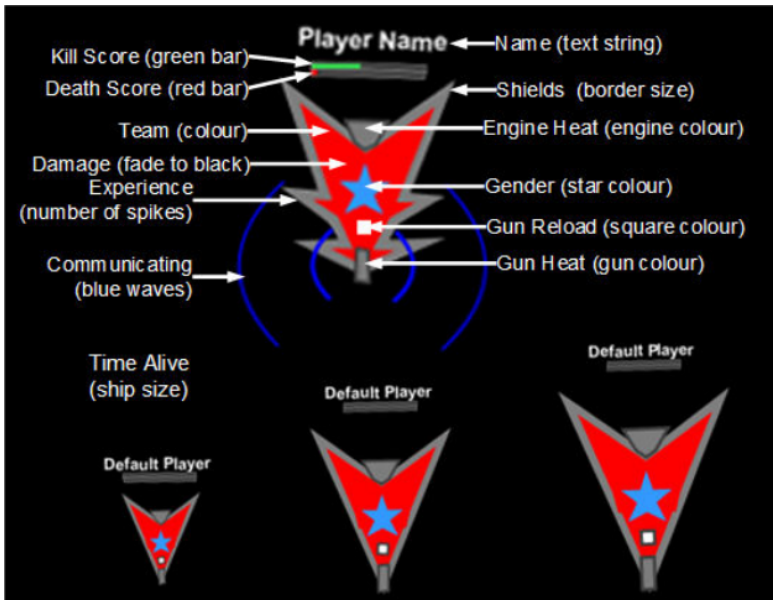


Figure 2.4: The information rich Spacewar avatar represents thirteen variables [36].

Rich Embodiment

In the real world there is much more contextual information available than what is represented in a typical embodiment, and *information-rich embodiments* have been developed to provide more of this contextual information in virtual environments [36]. Utilizing visual glyph characteristics such as color saturation, shape, size, icons, text, and bar graphs, these rich embodiments are able to represent many additional variables.

These types of iconic visual representations have been explored in previous work by Stach [36]. The first rich embodiment developed by Stach was an avatar type embodiment for a multi-player space combat game, Spacewar. Thirteen variables were represented on this embodiment including name, gender, game experience, team, kills, deaths, time alive, communication status, damage, shield strength, gun reload, engine heat, and gun heat (see Figure 2.4).

The second information rich embodiment developed by Stach was a telepointer type embodiment that was used in a shared drawing program [36]. In this example ten variables were represented which included name, age, gender, user color, activity

level, tool, brush color, stroke size, click status, and communication status (see Figure 2.5).

Commercial games have been using limited types of rich embodiment for years, in addition to the previous examples of Spore and Fable II (see Figure 2.6). Even simple games such as Super Mario Brothers for the Nintendo Entertainment System (www.nintendo.com) used size and color to represent information about the player's status. Interstate '76 (www.activision.com) is a car combat game that used shape, texture, and rendered smoke to indicate subcomponent damage (such as dents and flat tires) while using bar charts to indicate overall vehicle damage. Also, different types of equipped weaponry would be rendered on the vehicle. Knights of the Old Republic (www.bioware.com) varied the complexion of a character's skin to reflect his or her ethical choices.

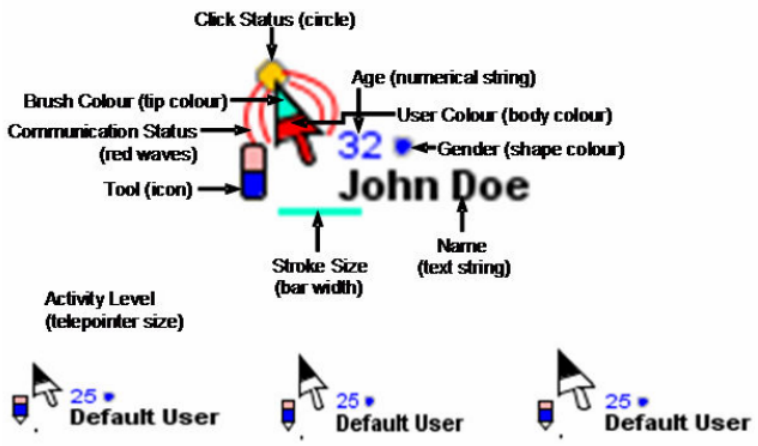


Figure 2.5: The information rich teleprompter embodiment represents ten variables. [36]

These rich embodiments were found to be feasible and valuable in groupware systems. People could remember and interpret large sets of variables, these variables helped people recognize and characterize others, and this additional information was used to enrich interaction.

Figure 2.6: Top left: Super Mario Brothers 3 for the Nintendo Entertainment System (www.nintendo.com). Top right: Knights of the Old Republic (www.bioware.com). Bottom: Interstate '76 (www.activision.com).

2.2 Information Visualization

Information visualization is the use of visual representations of abstract data to amplify cognition [7, 35]. This shifts the burden of understanding data from the cognitive to perceptive faculties and eases comprehension by "using vision to think" [7]. Three issues in the design of information visualizations are space, interaction, and representation. One aspect of information visualization is the use of physical space to take advantage of human skill in perception [7]. Computer assisted visualization techniques allow designers to take advantage of different interaction techniques which can support large data sets and shift effort from the user to the machine [7]. Information visualization also takes advantage of many types of symbolic representation to encode different types of numerical, ordinal, or categorical data [35].

2.2.1 Space

The use of space is one of the fundamental strategies in information visualization. There are four ways that space is used: visual structure, orthogonal axes, many complex axes, and using trees or networks. Visual structures are used to mirror the world and represent physical data. An example of using visual structure is the use of medical imaging to model the human heart [2]. Using the orthogonal axes of space can be very effective and can be done by using a single dimension [15], two dimensions [1], or all three spatial dimensions [4]. When visualizing data that contains more than three variables, orthogonal axes cannot be used, and some type of complex representation using multiple axes must be used [14]. A set of nodes, links, and associated information can be used to visualize a tree which often encodes hierarchical data [27] or a network which allows for cycles to exist [20] (see Figure 2.7).

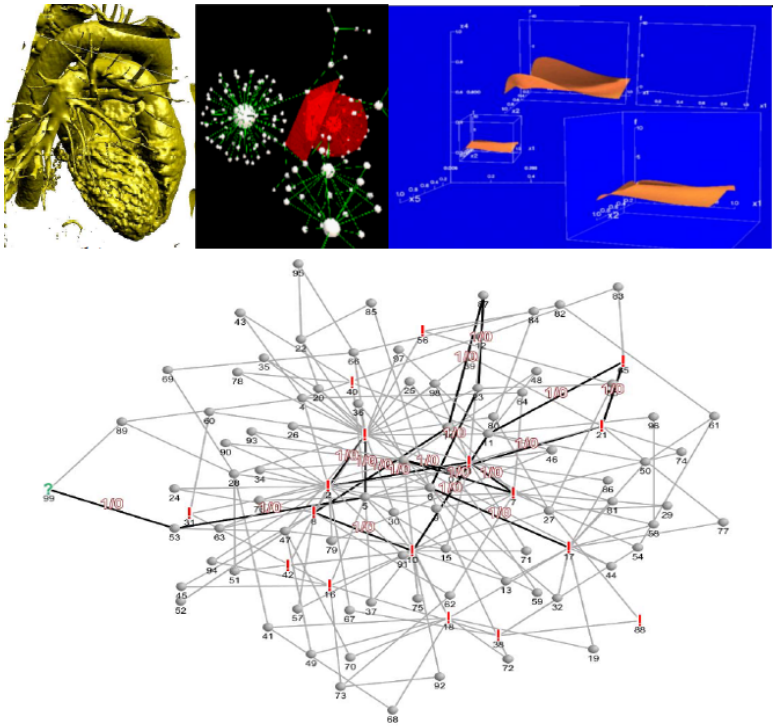


Figure 2.7: Top-left: Visual structure [2]; Top-center: 3-dimensional [4]; Top-right: multiple axes [14]; Bottom: networks [20].

2.2.2 Interaction

Computer-assisted visualizations allow for interaction to allow larger data sets to be managed and the effects of any modifications can be viewed in real time. Methods of providing interaction to visualizations include dynamic queries, interactive analysis, and overview plus detail. Dynamic queries allow users to manipulate visual parameter controls to rapidly generate queries to a database that return visual and animated results [34]. Interactive analysis extends dynamic queries by allowing more detailed comparisons, analysis, and precise, quantitative comparisons [8]. Overview plus detail is a technique that allows for massive scales of data to be handled at once. An overview is essential for navigation and detection of large patterns but users still require detailed access to data through the detail view; multiple views allows overview plus detail to handle datasets too large for any static method [32].

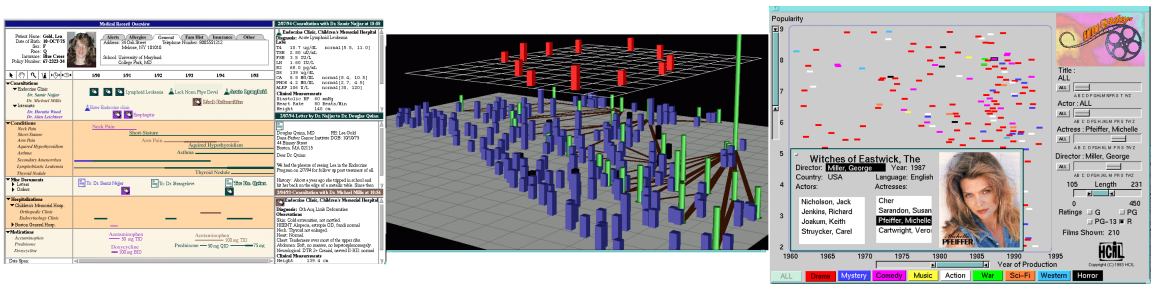


Figure 2.8: From left to right: Overview plus detail [32], interactive analysis [8], and dynamic query [34].

2.2.3 Representation

Information visualization uses representation to visually encode data within some design element of the visualization [35]. There are many elements of design which can be exploited to encode data, including size, spatiality, color, value, orientation, shape, and texture. Different representations are more or less suited to representing numerical, ordinal, or categorical data [35].

Data that is associated with a single number or value is numerical data and is

among the most common type of data to be represented. These require a representation with a large range and fine distinctions. Ordinal data is composed of elements that have some type of linear relationship or ordering that must be conveyed. Categorical data is a discrete category or class that a data element is a member of [35].

Size is the use of the area, length, or width of an object to represent a data element [9]. Humans have excellent spatial skills which can be exploited to easily represent ordinal or categorical data through the relative positioning and location of items such as the urgency or type of documents on a desk [25]. *Color* has long been used to represent different variables and many traditionally-used color sequences are well recognized by users through their real-world analogs or long standing use in areas such as charts and geographical maps [24]. *Value* is used by Bertin [5] as the relative darkness or lightness of a color. The *orientation* of elements within an area can also be used to distinguish between them also as illustrated by Bertin [5].

2.3 Parametric Texture Generation

Texture is a set of visual properties that most people will recognize intuitively, but can be difficult to define. Bertin defines textures simply as the fineness or coarseness of constituents in a given area while maintaining a constant color value [5]. There are many potential statistical properties of textures. Attempts to automatically characterize textures using human visual perception have yielded characteristics such as the shape of the texture primitives (also known as "textons"), "linelikeness" (how closely the textons resemble lines), coarseness, directionality (distribution of the orientation of individual textons), regularity (placement variation), contrast, and texton edge softness [18].

Procedural texture generation algorithms are controlled by a set of input parameters that change the resultant texture. If parameter changes create texture changes that are perceptible to users, these algorithms can be used for information visualization. There are many parametric texture generation methods that can provide a library of appealing, unique, or naturalistic textures [12]. These techniques are in

contrast to nonparametric texture generation methods which attempt to synthesize additional texture from an existing swatch and since nonparametric methods lack any controllable parameters, they are unsuitable for information visualization.

2.3.1 Perlin Noise

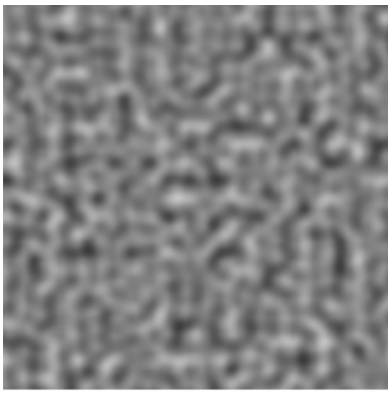


Figure 2.9: Perlin Noise. [12]

Perlin noise is named after its author Ken Perlin, and is a widely used method of procedurally generating a variety of graphical effects [29, 31]. This noise is used as a basis function which is itself uncompelling (see Figure 2.9), but is often combined with two- or three-dimensional texture generation methods to add a stochastic component to generated textures [29].

Perlin noise was introduced as a tool for image synthesis by Ken Perlin [29] where natural visual complexity is built up by composition of nonlinear functions to create convincing representations of clouds, fire, water, stars, marble, wood, rock, soap films and crystal.

The noise function was designed to provide two characteristics that are very useful when generating textures: reproducibility and smoothness. When evaluated with the same input parameters, the output will always be the same. This reproducibility allows a very compact representation of a texture to be saved and the texture itself to be generated as required at run-time. Smoothness is a product of smooth interpolation between underlying lattice points which allows the texture to be sampled at

any location and scaled to provide a texture with no jarring discontinuities or hard edges.

The noise function can be evaluated at any three-dimensional point. The algorithm divides space into a lattice of cubical cells and each lattice point is assigned a pseudorandom wavelet. A wavelet is a function that integrates to zero, drops off to zero outside of a certain region or radius, and square integrates to one. The wavelets used in the noise function have a radius equal to the distance between lattice points so that any given lattice point will only affect points contained within the adjacent 26 cubical cells. Wavelets are also designed to have zero at their center, a random gradient in their center, and smoothly drop off to zero. For any three-dimensional point the algorithm will calculate which cell the coordinates are in, compute the wavelet for each of the closest 26 lattice points, and sum the values [12].

An example of using the noise function to generate texture is the application of the noise function used to generate a realistic marble texture [29]. The appearance of marble can be generated by first color filtering a sine wave to create a series of color bands in the desired shades:

```
function boring_marble(point)
    x = point[1]
    return marble_color(sin(x))
```

The x value of the input coordinates (*point*) is used to generate a sine wave. The sine wave is converted into color through the *marble_color* function (see left panel of Figure reffig:perlinmarble).

A *turbulence* function is used to simulate turbulence from the Noise function:

```
function turbulence(p)
    t = 0
    scale = 1
    while (scale > pixelsize)
        t += abs(Noise(p/scale) * scale)
        scale /= 2
    return t
```

This *turbulence* function can then be used to add a stochastic element to the rendering of the marble which greatly increases its realism (see right panel of Figure 2.10).

function marble(*point*)
    $x = point[1] + \text{turbulence}(point)$
    return marble_color(sin(x))

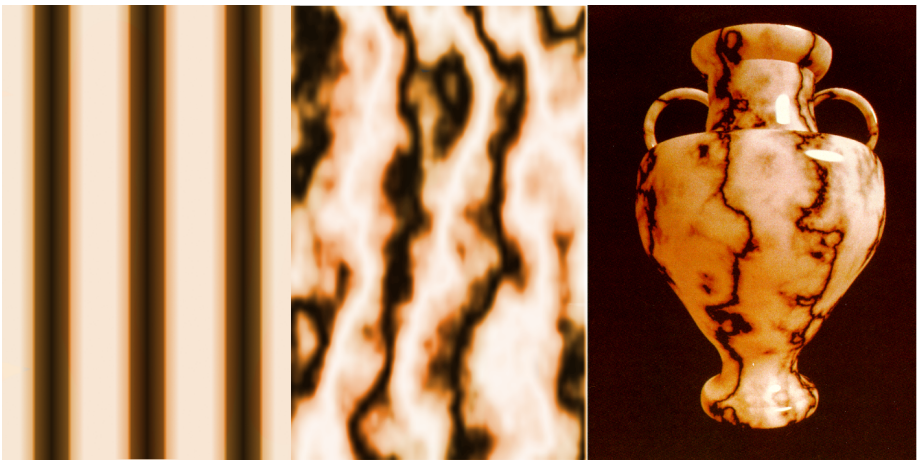


Figure 2.10: Left: Simple marble without added turbulence. Middle: Marble with turbulence. Right: An image of a vase textured with Perlin marble. [29]

More recently, the Noise function has been applied to generate other types of texture such as three-dimensional shapes for hair, fur, fire, glass, fluid flow, and erosion effects [31]. There was also an improved version of the Noise function developed to fix some minor problems with the original algorithm [30]. Problems addressed included discontinuities when using certain derivatives and a directional bias between the axis and diagonal directions on the cubic lattice.

2.3.2 Worley Noise

Worley Noise is a basis function that is similar to Perlin Noise although its application is not as broad. It is also known as *Cellular Texture* [43] due to the distinctive cellular patterns it typically generates. It is most useful for its ability to generate naturalistic textures that require a cellular component such as skin, scales, flagstones,

cracked mud or ice, or crumpled paper. The simplest type of Worley Noise has the appearance of a Voronoi diagram, and can be used as a basis to generate a variety of interesting textures.

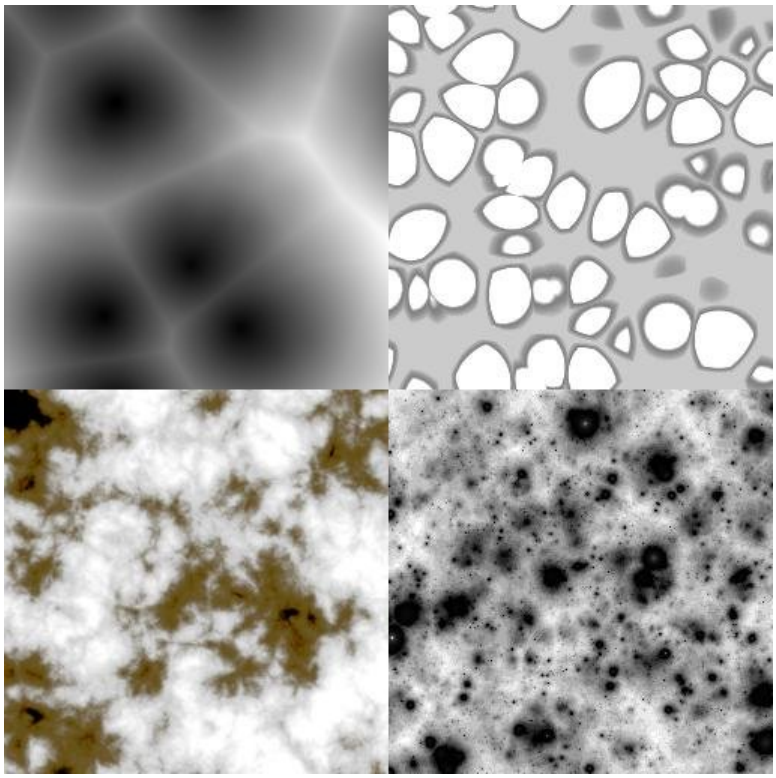


Figure 2.11: Several cellular textures. Clockwise from top-left: basis function F_1, basic cobblestone texture, cratered surface texture, rust texture.

Worley noise is also named after its original author Steven Worley as presented as a cellular texture basis function [43], and is generated using random placement of *feature points*. Feature points are randomly distributed throughout space (three-dimensions in Worley's original paper), and then for each point through space there is a feature point that it is closest to.

A function, F_1, is defined as the distance from each point to the nearest feature point. F_1 varies continuously throughout space although lines equidistant from two nearby points will form a Voronoi diagram of the feature points. Additional functions are defined as F_n that are computed to find the n-th closest feature point to any given point.

F_n can be evaluated for any three-dimensional point, and the first step in evaluating the function is to determine the location of nearby feature points. The distribution of the feature points is important to avoid introducing any artifacts so the feature points are spread throughout space using a Poisson distribution. Space is divided into cubes which each contain zero, one, or more feature points. These points must be placed randomly but reproducibly throughout each cube. A common approach is to use a hash function on the cube coordinates to initialize a fast random number generator which is then used to generate both the number and location of the feature points.

As function points are generated the distance between them and the original evaluation position is calculated and stored in a sorted list. The list only has to be large enough to hold as many values as the largest n value that is ever evaluated (typically between 2 and 4) and so sorting the list is fast and often performed with only a few hard-coded tests. Some neighboring cubes may also have to be checked for feature points, but most can usually be excluded with a simple bounds check. The entire sorted list is usually returned with each value corresponding to a value of n.

Different textures are produced from each n and linear combinations of the different functions are used to create interesting textures. Different scales and combinations of these functions provide a large set of texture possibilities (see Figure 2.11). Also, different distance metrics can be used to alter the appearance of textures. Using Manhattan distance will produce textures with square and rectangular features visible.

2.3.3 Reaction-Diffusion

Another method of generating texture is known as *Reaction-Diffusion*. A Reaction-Diffusion texture is generated by modeling two or more chemicals as they interact on a surface, and is based upon the way that certain textures, like the coats of certain animals, are generated in nature.

This method was original proposed by Turing in 1952 [41] where he showed

how two or more chemicals can react across a surface to form a stable pattern. For each chemical in the simulation there is a function that provides the increase or decrease of the chemical given the relative concentrations of all chemicals in that area which provides the *reaction* component. A second term in the equation provides the diffusion rate of the chemical given the nearby chemical concentrations which provides the *diffusion* component. A variety of different equations can be used, but with the right equations and parameter inputs Reaction-Diffusion can produce spots, stripes, and other patterns reminiscent of animal patterns in nature.

The general form of the equations for a two-chemical (chemical a and chemical b) Reaction-Diffusion system are:

$$\frac{\partial a}{\partial t} = F(a, b) + D_a \nabla^2 a \tag{2.1}$$

$$\frac{\partial b}{\partial t} = G(a, b) + D_b \nabla^2 b \tag{2.2}$$

The change of the concentration of chemicals a and b at any given time is determined by the sum of a reaction function (F or G) of the local concentrations of a and b and the diffusion of each chemical to and from places nearby. Chemicals diffuse from locations of high concentration to nearby areas of low concentration [42].

The range of textures possible can be increased using a cascade process of Reaction-Diffusion systems where patterns are created by one system and then refined by a different system. This was demonstrated by Turk [42] who also demonstrated a method of generation Reaction-Diffusion directly on 3D models (see Figure 2.12).

2.3.4 Texture Splatting

Texture splatting is a texture generation method that creates a texture by taking a set of texture primitives, or "textons", and stamps or "splats" them into a texture. Successive and repeated copies of the texture are placed until the entire texture is sufficiently filled. The advantage of texture splatting is the degree of control over the placement of the textons such that texture statistics (such as the frequency of

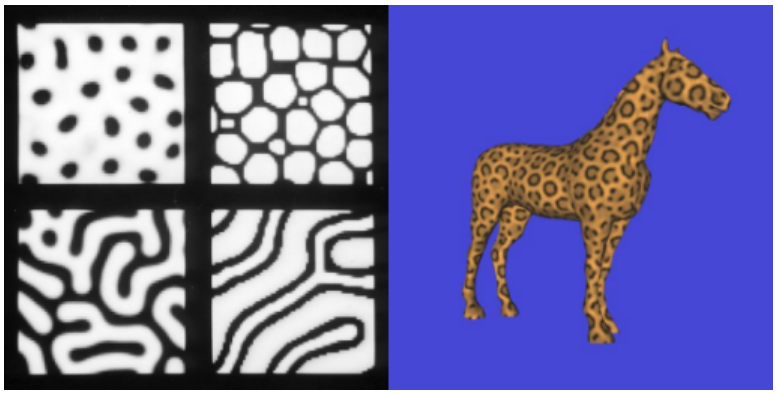


Figure 2.12: Left: Example of patterns generated by Reaction-Diffusion. Right: Texture generated directly on horse model. [42]

texture features) can be fine-tuned. This type of control allows this type of texture generation method to be used in texture synthesis algorithms that generate arbitrary amounts of a new texture through matching these statistics to a seed texture [16].

More recent techniques for texture splatting allow sprites to be applied to three-dimensional objects in real-time [21]. New types of surface aspects that would be difficult to create with other methods can be edited and animated interactively.

2.4 Procedural Shape Generation

Generating shapes procedurally involves any method that uses a set of rules that are algorithmically applied to produce a certain type of structure. Among the possible methods for accomplishing this are shape grammars and L-systems. Procedural shape generation relies on the self-similarity or fractal nature of certain structure and has been used very successfully on objects such as plants, trees, and buildings.

2.4.1 Shape Grammars

Shape grammars are a recursive shape computation that uses shapes as the primitive and applies shape-specific rules; the technique was originally presented by Stiny as a "complete specification of families of non-representational, geometric paintings and sculptures" [38]. Shape grammars begin with an initial shape and then recursively

apply the shape rules. The left side of each rule is matched to a shape and is then substituted for the right hand of the rule (see Figure 2.13). Any geometric transformations (scale, rotation, translation, or mirror image) can be applied to any shape replacement rule as long as it is applied to both sides of the rule–any replacement shape will be moved, scaled, or rotated to match the shape it is replacing. Rules are applied sequentially, one at a time, and usually continue until there are no shapes that match any rule. Shape grammars focus on the shapes themselves as primitives which makes them useful in areas such as describing, analyzing, and generating both paintings [19] and architectural designs [6].

Shape grammars can also be used to automatically create visually distinctive shapes or icons such as VisualIDs [22]. VisualIDs are generated to provide memorable, unique visual identifiers to take advantage of human visual memory to aid navigation through a file system (see Figure 2.14).

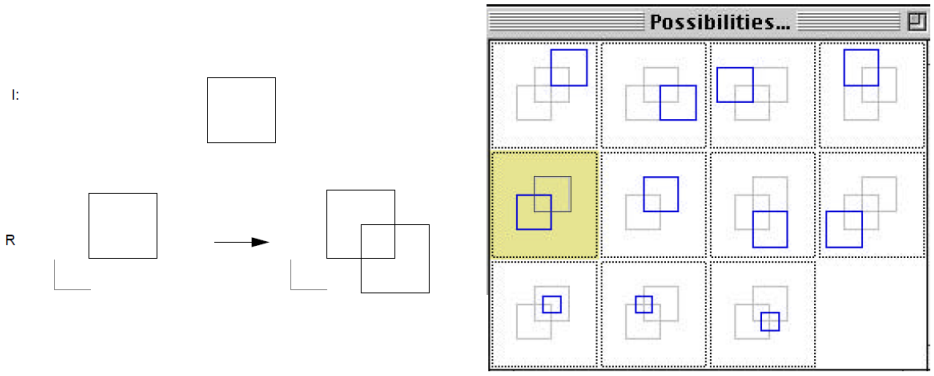


Figure 2.13: Example of a simple shape grammar and some of its possible applications [40].

2.4.2 L-Systems

L-systems are a mathematical formalism introduced by Aristid Lindenmayer in 1968 [23] as an axiomatic theory to model cellular development. L-systems work by applying a set of rewriting rules, most commonly on a character string. A set of replacement rules or productions are successively applied to a given string to generate more complex objects. These tokens require a geometric interpretation that

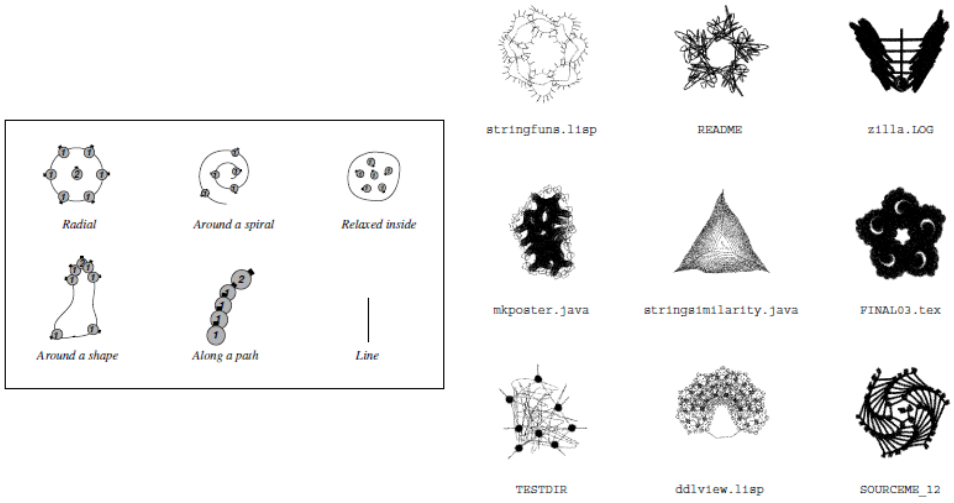


Figure 2.14: Example of VisualID initial conditions (left) and examples of generated icons (right) [22].

allows them to be rendered into a shape. A common interpretation is to use the tokens as commands for a two- or three-dimensional "turtle" that has the ability to draw a segment, rotate left/right, roll left/right, pitch up/down, save/restore state, or perform other maneuvers. The L-system to create a Koch snowflake [12] has only a single rule and three tokens which are interpreted as commands to a two-dimensional turtle (see Figure 2.15).

Unlike shape grammars, the rules of an L-system are applied in parallel to every token at the same time, as the original motivation was to model the behavior of cellular division [26]. L-systems have become popular in computer graphics for modeling realistic plants [33] and generating fractals [10] (see Figure 2.16).

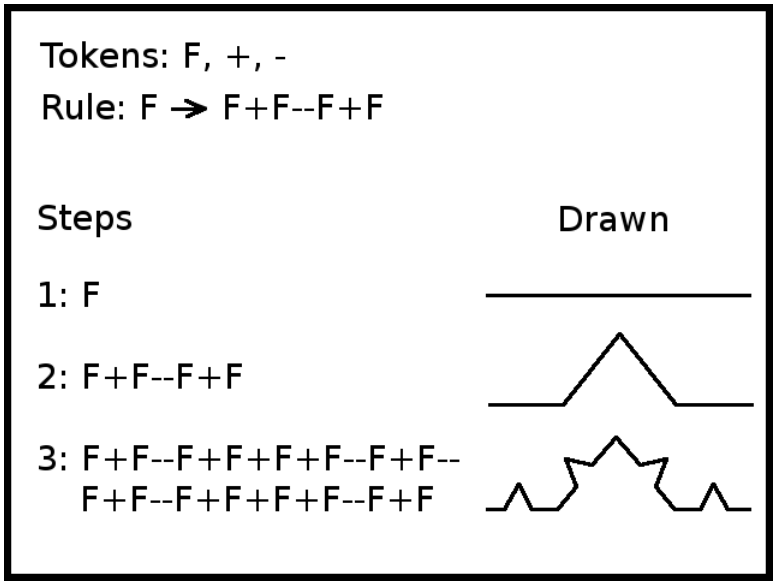


Figure 2.15: Example of a simple L-system for a Koch snowflake. The tokens are interpreted as two-dimensional turtle commands. "F" means draw a forward line segment, "+" means rotate left, and "-" means to rotate right [12].

Figure 2.16: Example of L-system generated images of plants [33] (above) and fractals [10] (below).

Chapter 3

Information through Texture and Shape

Parametric texture generation and generative shape grammars provide adjustable parameters that can be used to produce visual effects that convey information on an avatar. This section explores two possible methods for creating parametric visual effects: texture and shape. Both these methods provide flexibility for creating distinctly recognizable visual features.

These adjustable parameters have a certain range in which they can be adjusted; this parameter space will correlate with some amount of visually perceptible difference in perception space. This relationship between parameters and perception can be used to represent information to users. The amount of parameter change required for a user to visually distinguish between two textures can be used to represent a distinctive value. The number of these distinct values that can be represented within a type of texture determines its usefulness for information visualization.

3.1 Parametric Texture Generation

There are many methods for generating textures, but not all are suitable for information visualization. There are a number of required properties a method must have. The primary required property is that the method allows input parameters to control the generation of textures. Another required property is for these parameters to be mapped to visually distinct properties that users can recognize. The ability to generate a sufficiently large variety of different textures is also important to fully express different types of data. However, even methods with a limited range of visual effects can still be effectively utilized by matching them to a similarly limited type

of data. Computational cost of any method is another property that is useful to consider but can generally be overcome by precomputing texture data. Performance can allow for more real time manipulation of textures to provide large amounts of texture and texture animation without requiring prohibitive amounts of memory for precomputed texture data.

3.1.1 Perlin Noise

Perlin noise is a very common method that is widely used for a number of applications including texture generation (see Section 2.3.1). Raw Perlin noise itself is visually uninteresting so it is commonly used as a two- or three-dimensional input into a texture function that determines the visual properties and parameters of a generated texture. Fortunately, the primary strength of Perlin-noise-based texture is the wide variety of textures possible when using it as a base. There has been significant work using Perlin noise to generate texture, so there are many readily available techniques to produce naturalistic textures of materials such as wood grain, marble, or clouds. At a minimum, Perlin-noise-based textures will have a parameter controlling the amount of visual noise introduced into the texture, but there will often be other parameters that control other visual properties of the textures such as the frequency and size of texture components and the applied color map. Finally, Perlin noise has many highly efficient generation algorithms so its performance can be excellent.

3.1.2 Worley Noise

The signature feature of Worley noise is the cell-like pattern that is evident in the base texture which allows for the generation of a unique set of textures (see Section 2.3.2). This distinctive feature is the greatest strength of Worley noise and allows it to easily generate textures that exhibit a cell-like structure such as cobblestones, scales, cracks in mud, or even a field of puffy clouds. Its strength is also one of its weaknesses in that it is not easily utilized when generating textures that do not exhibit a cell-like structure. Worley-noise-based textures have many adjustable

parameters including scale, frequency, and contribution from each F_n, and also have texture-specific parameters. The performance is also efficient enough that real-time computation within modern graphics processing units (GPUs) is possible.

3.1.3 Texture Splatting

Texture splatting is a simple method and its greatest strength is the amount of control possible when creating texture with it. The direction of individual textons can be finely controlled to add anisotropy (directional dependency) to a texture. The spacing and size can be easily controlled to produce textures with a different density and regularity while individual texton shape can be changed to give the texture a different appearance. All of these elements are visually distinct features that have been widely used for information visualization purposes. The performance of texture splatting is limited by the number of textons required for a desired texture, but with modern advances in both GPU power and efficient algorithms it is possible to do texture splatting on three-dimensional objects in real time. There are specific types of textures that may more easily or efficiently be produced using Perlin or Worley noise but it provides excellent control of the distribution and features of the textons.

3.2 Texture Library

I developed a Java application to help explore and experiment with parameters for both Perlin and Worley Noise. The appearance of textures generated with different parameters are sometimes difficult to predict or control, so this tool provided a method of exploring the texture space and getting an intuitive impression of the types of textures available.

The application initially displays a window providing the set of current textures and color maps (Figure 3.1). Texture generators and color maps are all objects that can easily be changed or swapped within the source code. Selecting a box fixes the texture and color map and opens a new window which allows you to select which two parameters of the texture you would like to alter (Figure 3.2). Selecting the

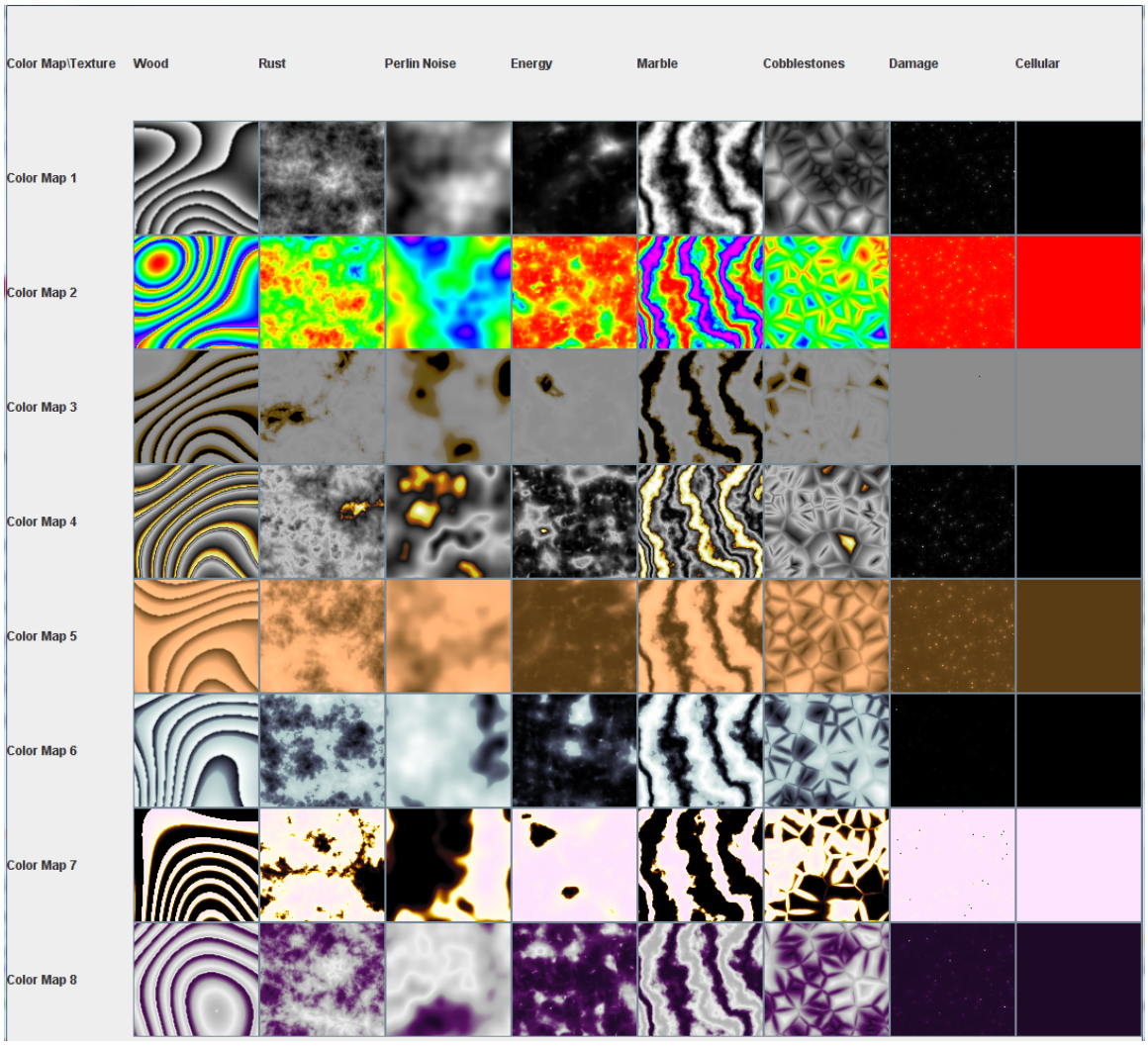


Figure 3.1: Texture library initial window.

parameters will open a new window generating texture swatches for a set of textures with the two selected parameters assigned to the axes (Figure 3.3). The range of the parameters can be altered by clicking on them and entering a new range. Selecting a box within this window will fix the two varying parameters, and allow the selection of two different parameters. Old windows remain open so it is easy to backtrack and try different options.

This tool aided in the development of different texture types, and was useful for determining a general understanding of the relationship between perception space and parameter space for a given texture. This relationship is of primary importance

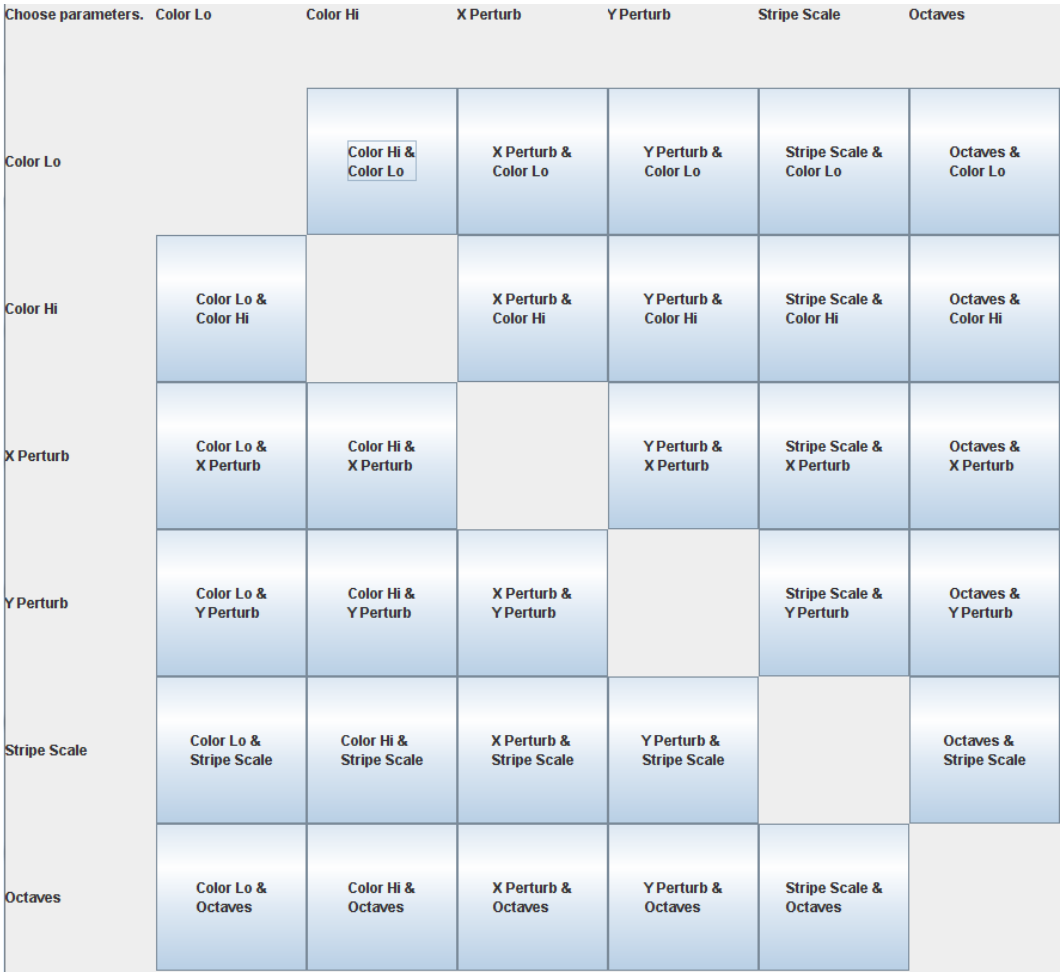


Figure 3.2: Selecting texture characteristics to manipulate.

when using textures to represent information and is further explored by the study in Chapter 4.

3.2.1 Parameter Space Versus Perception Space

Parameter space is the range of actual values used in a texture generation algorithm. Depending on the texture, this space can have multiple dimensions which each alter the generated texture. For the texture types generated for this thesis each focuses on a specific characteristic, and so parameter space has been simplified to a single dimension. Each texture characteristic is determined by a function of the texture parameters to only alter the texture characteristic being altered. For example, the implementation of the "Turbulence" texture (described later in this chapter) has

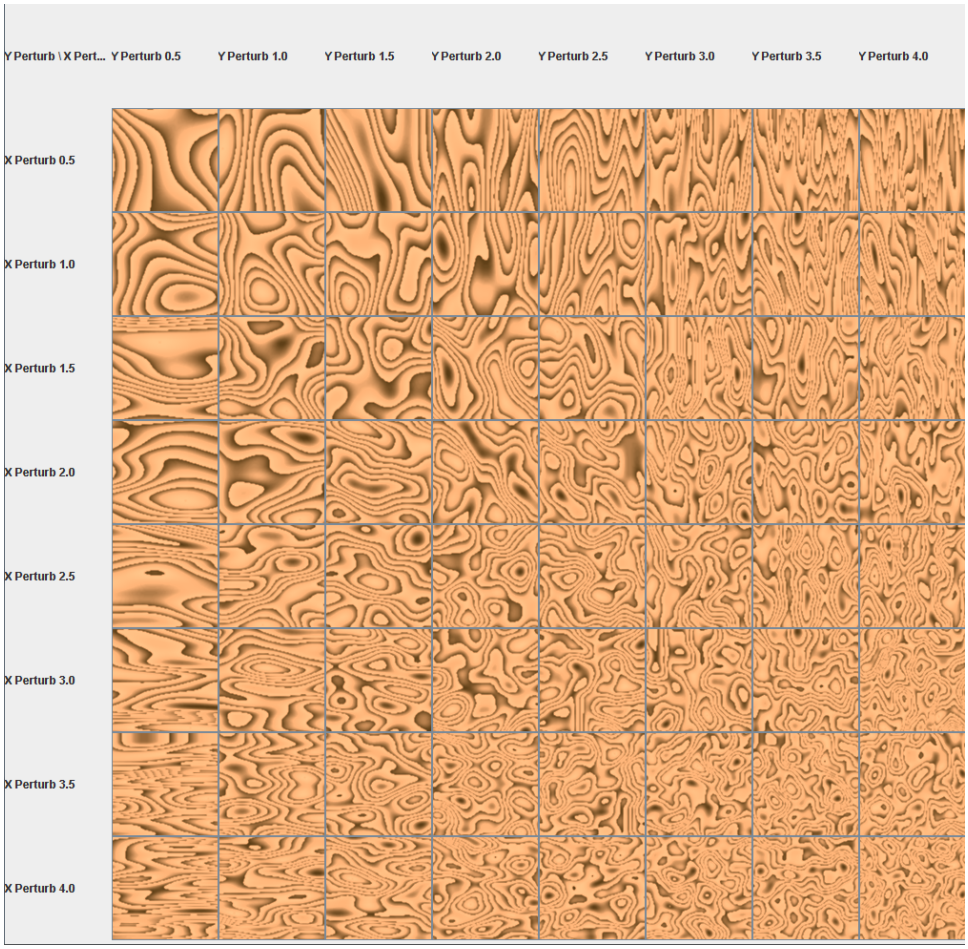


Figure 3.3: Example of varying texture parameters. This example shows the amount of turbulence in each direction.

separate input parameters for the X and Y axes, but a single parameter is used to generate the Turbulence texture, which affects both axes the same amount, as well as mapping onto all other parameters for the Turbulence texture. Parameter space is large with great amounts of variation that may not be meaningful to a human user so these parameter functions were designed manually to isolate interesting texture features.

Perception space is used in this thesis as an abstraction for a measure of perceptible change within a texture. Many different textures can be generated, but it is important to determine what textures are perceptibly different to a human observer. For example, two different textures full of static could be completely different on a pixel-by-pixel basis but be virtually indistinguishable to a user. On the other hand,

even a slight modification to a texture of very regular lines will be quickly noticed by a user even though the majority of the pixels remain identical.

Human visual perception is complex, but for the purposes of using texture to represent information we are only interested in its relationship to the texture parameters we have control over. By using a single parameter input for the texture generation function, perception space can be simplified by only considering the amount of perceptual change for a given texture parameter value. For any texture generation parameter there is a certain amount it must be changed before a user can perceive a difference. An understanding of this relationship allows one to know how much to alter an input parameter to appropriately represent a particular value. This allows information to be successfully presented to user without requiring a full understanding of the underlying processed governing human perception.

3.3 Texture Characteristics

To develop a baseline understanding of the appropriateness of texture as a vehicle to convey information in a groupware system, a set of texture characteristics were chosen. They were selected to cover a broad range of possible methods for manipulating texture although they are by no means comprehensive. For each texture characteristic identified, a texture was chosen which exemplifies the texture characteristic. For example, the Turbulence texture characteristic uses a marble texture which demonstrates the characteristic.

Each texture was developed with a single parameter input value used to control the appearance of the texture and the precise details of the generation are abstracted out to improve the analysis of the results. The focus of these texture characteristics is to determine what the potential for texture in general would be, and what texture characteristics are the most useful for future consideration.

All texture were designed to be grayscale only. Color is itself an entire dimension of variability, and these initial texture characteristics were chosen to only alter luminance. Once the feasibility of raw texture by itself is determined, color can be

added to enhance and extend the expressive range of any given texture.

An example of the texture and how they change with their parameter value is given in Figure 3.4. It should be noted that some characteristics use a negative property, such as sparsity, instead of a positive property, such as density. An increase in sparsity is really a reduction of density. This choice was made because in an early pilot study most textures were found to change appearance most rapidly at one extreme edge of their parameter range. For example, it is very easy to detect a slight change of regularity within a very uniform texture, but a slight change of regularity within a highly irregular texture will go completely unnoticed. This area of rapid change was chosen to correspond to low parameter values to increase the effectiveness of capturing these areas in the study described in Chapter 4 and to make it easier to compare the results from different texture characteristics.

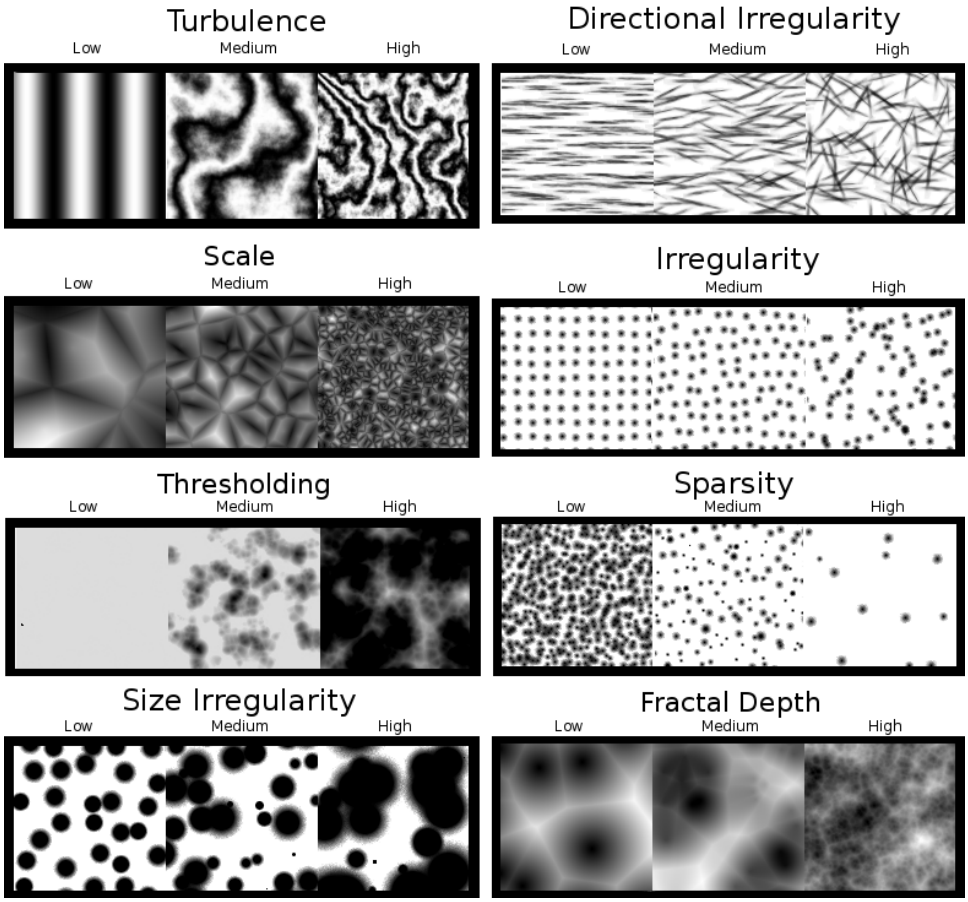


Figure 3.4: Examples of texture characteristic swatches.

3.3.1 Turbulence

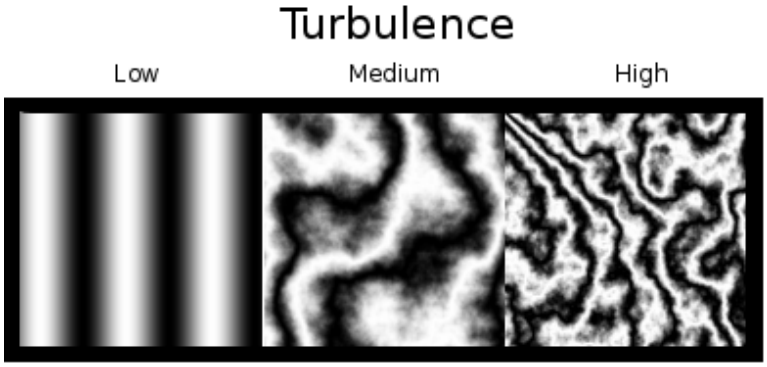


Figure 3.5: Example of Turbulence.

Turbulence is a texture characteristic that is based on Perlin noise (2.3.1). Turbulence in this case is being applied to the well-known Perlin marble texture, and scales the amount of turbulence (see Section 2.3.1) introduced into the texture. With the lowest value of turbulence the texture is merely a set of vertical lines. As turbulence is increased, the boundaries between the lines become more and more jagged as more noise is introduced into the texture.

The equation used was: $sin(y/StripeSize + Noise(x, y) * Turbulence)$ where x and y are the texture coordinates, $PerlinNoise()$ is the function that returns the value of Perlin noise at a given coordinate, $StripeSize$ is the size of the marble stripes, and $Turbulence$ is the amount of turbulence introduced into the texture.

3.3.2 Fractal Depth

Fractal depth is a texture characteristic that combines multiple self-similar copies of a Worley Noise texture at different scales of itself. The base texture used is simply F_1 of Worley noise. At low parameter values the texture is F_1 alone which is simply a shaded Voronoi diagram (see Figure 3.6). As the parameter is increased the visual complexity of the texture is increased by progressively blending in the same texture at a smaller scale. There are a maximum of six octaves of texture combined and each additional octave is scaled down by half in both size and intensity. Throughout the

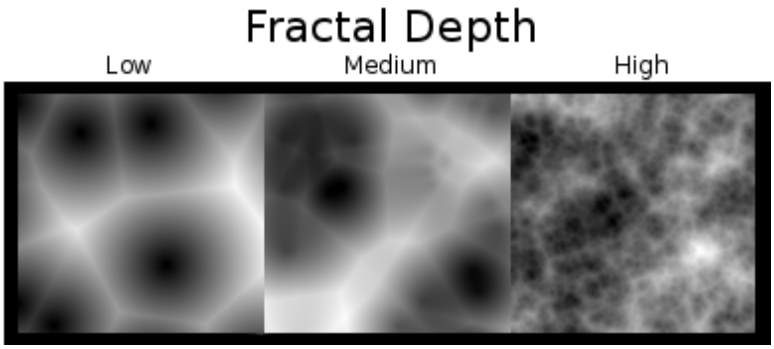


Figure 3.6: Example of Fractal Depth.

parameter space an additional octave is linearly blended every 20% of the parameter space. For example, at a parameter value of 10% the second octave of texture is blended at 50% opacity, and when the parameter value is at 40%, there are three octaves fully blended together.

3.3.3 Thresholding

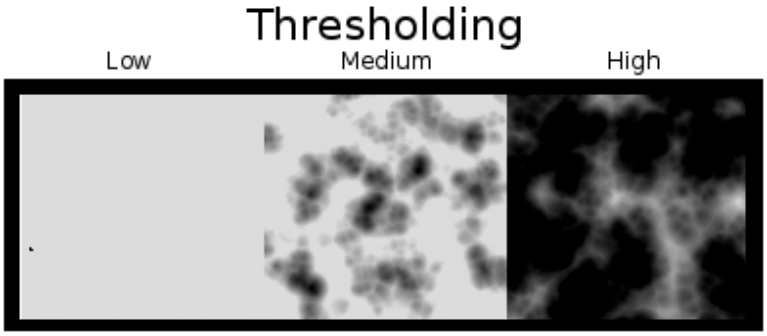


Figure 3.7: Example of Thresholding.

Thresholding is a characteristic that involves only manipulating the color map applied to the texture, and the texture used was four octaves of Worley noise (with each additional octave having half the scale and intensity of the previous octave as was the case with Fractal Depth). In its initial state the texture is entirely light gray, and in its final state the texture is completely black. The transition is split into two equal halves. Initially all color values are thresholded to the maximum value of light grey. As the thresholding level is increased, the darkest areas of the

texture are revealed. Through the first half of the parameter space, the thresholding level is linearly increased until the entire texture is displayed. The second half of the parameter space begins linearly thresholding the texture values to absolute black starting with the darkest areas and continuing until the entire texture is black.

3.3.4 Sparsity

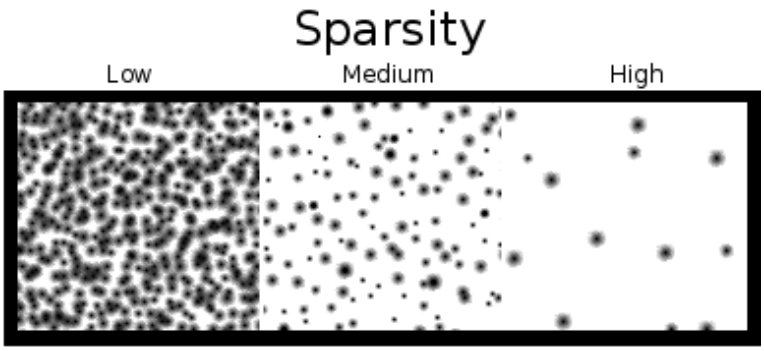


Figure 3.8: Example of Sparsity.

The sparsity characteristic used the texture splatting technique which allows exact control of the spacing of the textons. The lowest parameter value has the textons so close that the entire texture is black, and as the parameter is increased the textons are moved apart until only a few appear in the texture. The textons are not completely uniform, as random jitter is added both to their placement and size.

3.3.5 Irregularity

The Irregularity characteristic uses the texture splatting technique to place simple, circular textons. With the lowest parameter the textons are placed in perfect rows, and the parameter is increased the textons are moved out of position by a random amount using a uniform distribution of random numbers. The maximum amount of movement for the textons is increased linearly with the parameter value.

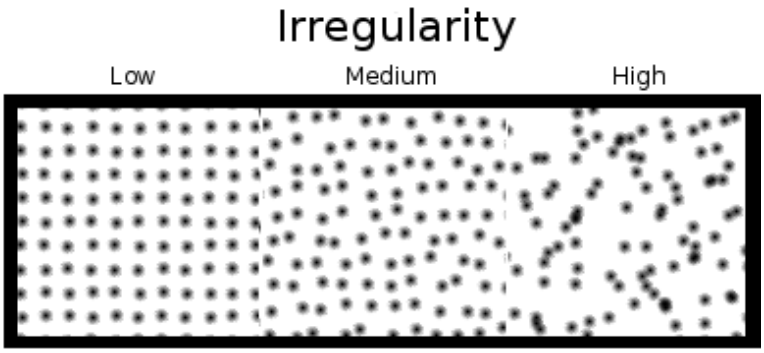


Figure 3.9: Example of Irregularity.

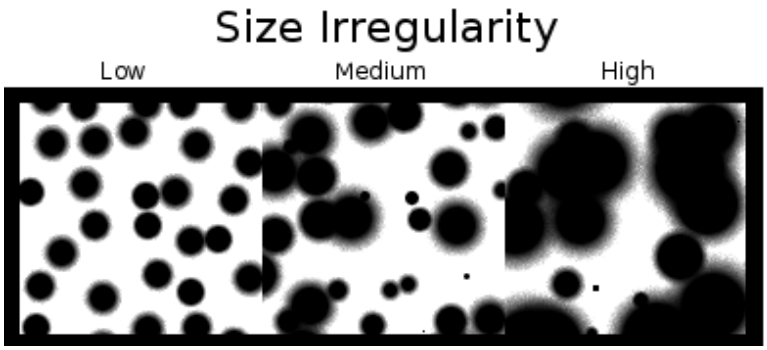


Figure 3.10: Example of Size Irregularity.

3.3.6 Size Irregularity

Size Irregularity is very similar to Irregularity except instead of distance between textons it deals with the radius of the textons. With the lowest parameter input the textons are all exactly the same size, and as the parameter is increased the textons are allowed to vary in size randomly using a uniform distribution of random numbers. The amount of variation possible is increased linearly with the value of the parameter.

3.3.7 Scale

Scale is a straightforward texture characteristic that simple alters the scale of the texture, and is generated using Worley noise. The combination used is the common "cobblestones" texture which is produced with (-1)*F_1 and (2)*F_2.

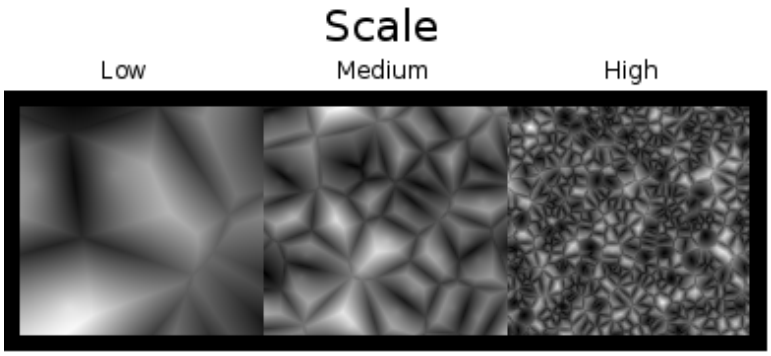


Figure 3.11: Example of Scale.

3.3.8 Directional Irregularity

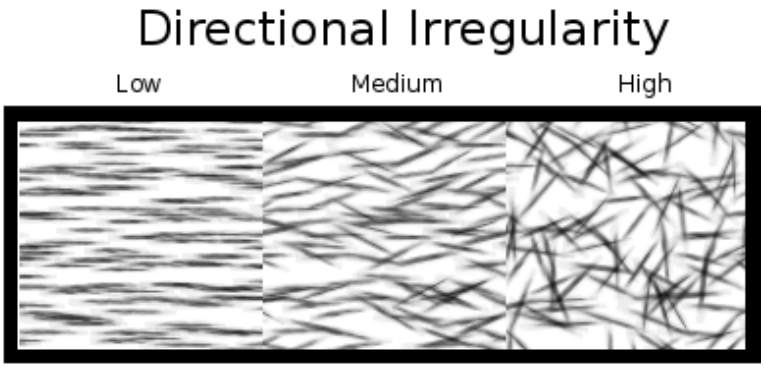


Figure 3.12: Example of Directional Irregularity.

The "Directional Irregularity" texture is generated using the texture splatting technique to lay down oblong-shaped textons. The initial state of the texture ensures that all textons are placed with the same orientation. As the parameter value increases, the textons' orientations are randomly shifted by an amount determined by the parameter value, and so Directional Irregularity could be considered the irregularity of texton direction.

The maximum rotation amount is increased linearly from 0 to 90 degrees throughout the range of the parameter. The amount of rotation is determined for each texton using uniformly distributed random numbers. The rotation can range anywhere from the maximum shift amount in a negative direction to the maximum shift amount in a positive direction. For example, when the Directional Irregularity parameter is at

50%, all textons will be randomly shifted between $-45°$ and $+45°$.

3.4 Texture Examples

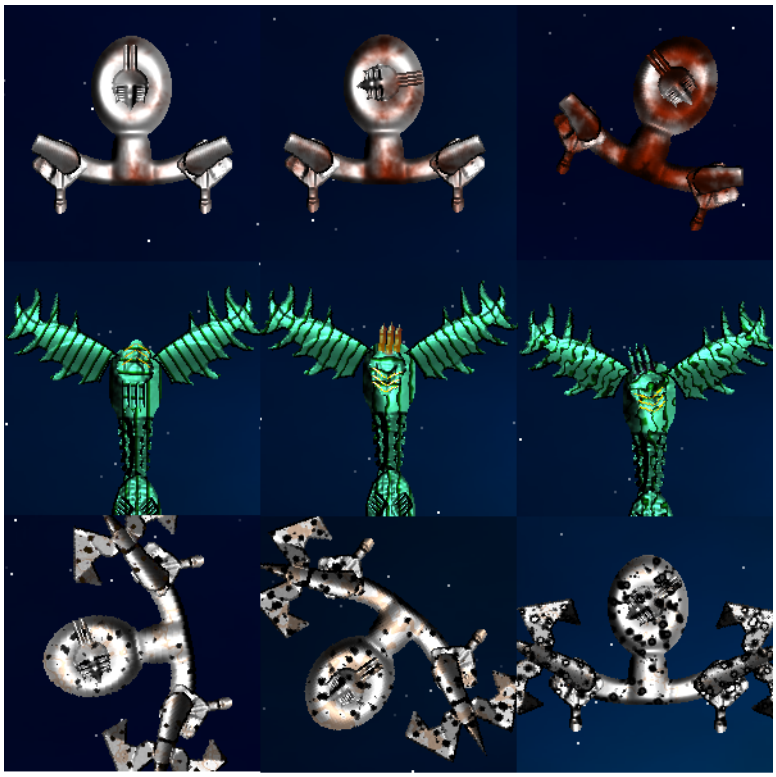


Figure 3.13: Example of texture use on spaceship avatars.

These textures can be applied to avatars and examples are provided in Figure 3.13. The top row is an example of a Thresholding texture, the middle row provides an example of a Turbulence texture, and the bottom shows a combination of a texture using Sparsity and Thresholding. These types of parametric texture generation techniques can produce very naturalistic, high quality textures.

3.5 Shape Generation

This section details an implementation of a shape grammar designed to provide a proof of concept for designing multi-level shape grammars that could potentially convey different types of information. A spaceship metaphor will be used to provide

an example of a shape system. A ship provides a useful example because it is a structure that includes a few distinct components including wings, guns, engine, and a hull, although these techniques could be applied to any shape.

There are two distinct levels of abstraction used in the generation algorithm: primary components that represent large-scale ship attributes, and subcomponents which are the graphical primitives used to build the shape of the primary components. The primary components are selected using a set of parallel replacement rules, and then the subcomponents can be applied to sequential replacement rules to selectively and incrementally increase the visual complexity of the shape.

The primary components are replaced in parallel which is a similar to the type of replacement within an L-system. All applicable rules are applied to all possible tokens during each replacement step. In this example, there is only a single parallel replacement step where all primary components tokens are replaced with applicable rules. The selection of these components would map to primary ship characteristics such as team, class, or abilities which is reflected in the style, shape, size, and type of primary components selected. This initial replacement could be random or the result of rule selection by a user.

The subcomponents of the ship are replaced serially in a manner which is similar to shape grammars. Each replacement step chooses a rule and then finds a single subcomponent on the ship that matches the rule. The rule is only applied to that subcomponent. This type of replacement can be controlled to reflect the state of the ship by selecting appropriate rules to apply. For example, a ship with upgraded engines would result in selection of several rules that apply only to engine subcomponents and these modifications would be obvious in the appearance of the engines. If its wings were never upgraded, no rules that altered wing subcomponents would be chosen so that the appearance of the wings would remain unchanged.

3.5.1 Primary Components

Primary components are large-scale features of ships that can represent the most important attributes. For the ship grammar, the large scale features include the

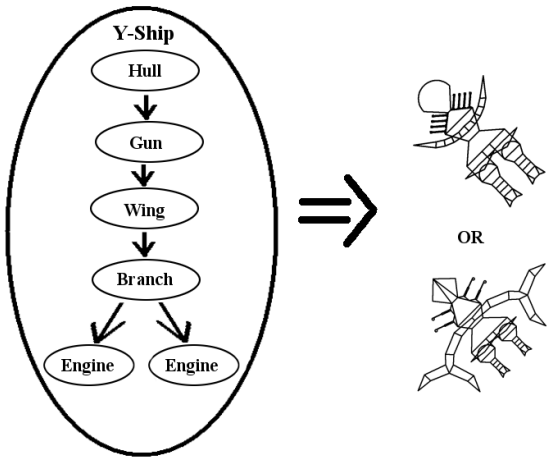


Figure 3.14: An example of one configuration of primary components: a "Y-Ship". This configuration consists of a hull segment, gun segment, wing segment, branch segment, and two engine segments.

different types of ship segments such as the cockpit, wings, engines, guns, or a hull section. The configuration of these components are simply organized in a tree (for an example, see Figure 3.14) of tokens representing a different type of component. Each token attaches to one or two other tokens of a specific type. The type and number of ship tokens identifies the class or type of the ship. For example, a class of ship called a "cruiser" could be identified by having a cockpit, gun segment, wing segment, and dual engines (indicating it has increased speed advantage over other potential ship classes). The basic characteristics of the ship can easily be identified by the type and number of segments it has. Examples of different ship configurations is provided in Figure 3.15.

A specific ship is created by applying a simple parallel replacement rule which replaces each token with a specific style of component. Different component styles could be used to represent certain characteristics of the ship or characteristics of the individual ship components. Distinct styles can also be used to recognize other players through their specific combination of components.

Additionally, these primary components have a few adjustable parameters that can be used to represent additional information. The size, proportion, and angles can be changed for all components. Some components have additional options such

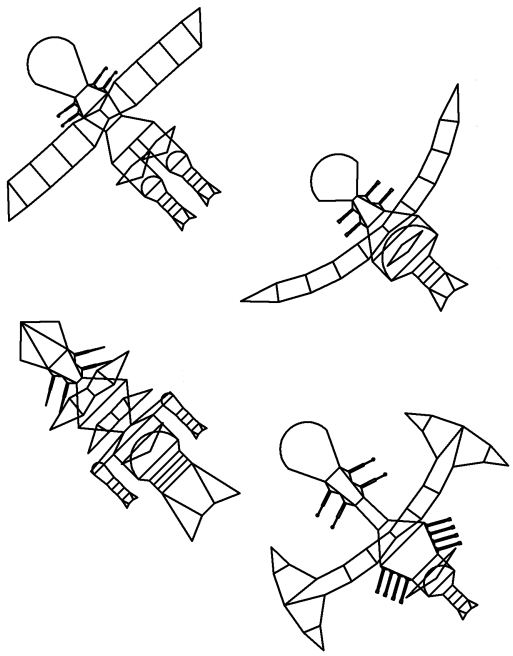


Figure 3.15: Examples of different configurations of ship primary components.

as the wing segment which can have different symmetry types or the gun segment which has a few different rendering styles for the cannons. These options could be used to represent more specific attributes of the ship such as the wing's forward or backward curvature which could indicate information such as the ship's relative ability in acceleration and handling. The size of the engines could be mapped to maximum speed while the ship acceleration could be represented by the number of engines.

3.5.2 Subcomponents

The second level of shape generation acts on the subcomponents or basic building blocks of the ship. The method for this is a simple L-system which sequentially replaces one component with multiple components. For most ships, a small set of subcomponents are often rendered multiple times to complete the entire ship. For example, there may be one hull segment that is used multiple times to outline the hull. This hull segment is itself composed of one or more simple line segments.

Replacement of a line segment within the hull segment with a zig-zag pattern would affect the entire outline of the ship. A hull could be designed using multiple hull segments (for different areas of the hull or multiple components alternating for the entire outline) and different replacement rules would act on each hull segment for a variety of visual effects.

Each individual line segment of a new zig-zag subcomponent would itself be added to the list of subcomponents to potentially be replaced in the future. Subsequent applications of the zig-zag operation could further alter the outline of the ship by replacing line segments that resulted from a previous application of the zig-zag operation.

There are also non-rendered "seed" subcomponents that allow the placement of upgrades on the ship such as communication dishes, missile bays, or other designed objects. Examples of different ships and applied effects are provided in Figure 3.17.

Using this technique the outline of the primary components could be altered to indicate age or damage with zig-zags or made thicker with subsequent layers of armor plating. The primary cannons of the gun segments could be made longer by adding additional segments, or large communication antenna could be created using multiple applications of the communication subcomponent.

Each replacement rule generally introduces a number of smaller components that replace a single larger component. Therefore, the number of subcomponents that can be added is limited by the ultimate size and resolution of the object being created. A replacement limit can be enforced to avoid rendering objects that are too small to see. Even if an object is still clearly visible, a greater number of subcomponent replacements causes newer components to be smaller and to appear less important.

3.5.3 Ship Examples

To illustrate the potential use of the ship generation algorithm this section provides an example. This example uses Figure 3.18 which provides an example "battle-cruiser" type vessel and compares it to another given "gunship" type vessel.

Using the example characteristics mentioned earlier I will demonstrate how these

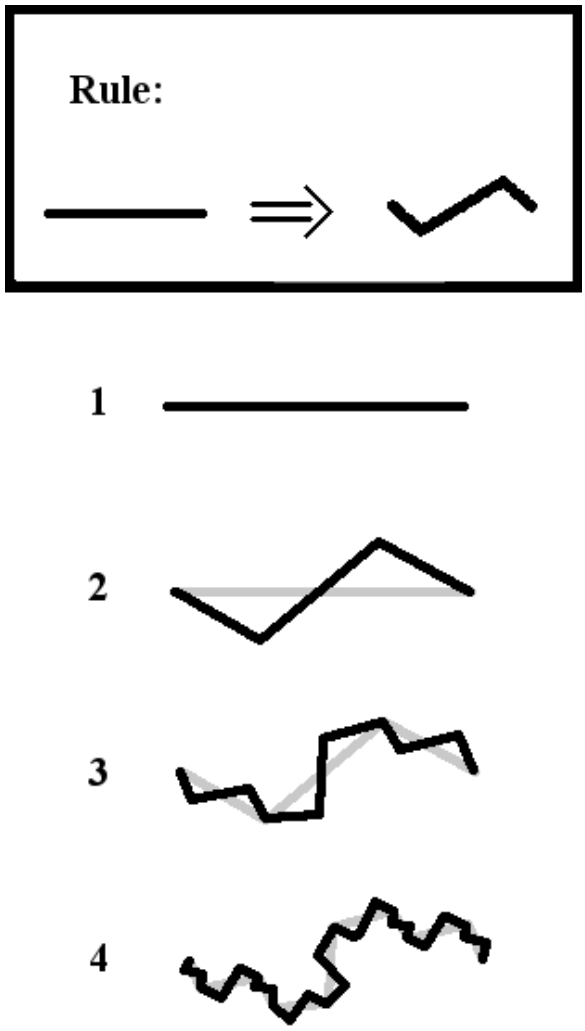


Figure 3.16: An example of a simple replacement rule and its application.

visual properties could be used. The gunship is quite thick which could indicate it is heavily armored, but a quick glance at the battle cruiser's guns indicates they are very large and likely sufficient to defeat the armor. The battle cruiser has relatively weaker armor so the gunship's many smaller guns may still be effective. The actual interplay of armor and weapon values represented would be game dependent, but an experienced player would be able to easily recognize the attributes of a ship by its visual appearance and how those attributes would affect the outcome of combat.

Another potential visual indicator is the number and size of the engines. The high

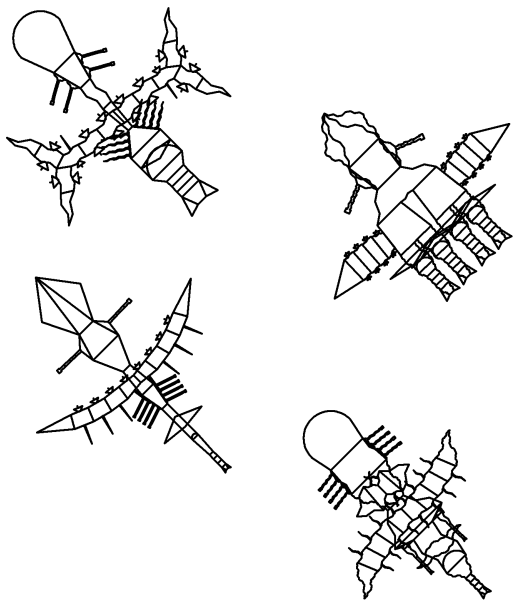


Figure 3.17: Examples of different alterations of ship subcomponents. Note the alteration of the outline of some ship components, and the addition of objects on the perimeter of the ships.

number of engines the battle cruiser has (four) indicates it can accelerate quickly and is maneuverable with its large wings so it can possibly out fly the gunship. However, the gunship's single engine is much larger than its opponent's so once it achieves its top speed it will be able to get away.

The example demonstrates how by glancing at the embodiments of other players a user can improve the complexity of interactions within the system. This additional information can inform and enrich the interactions of all users.

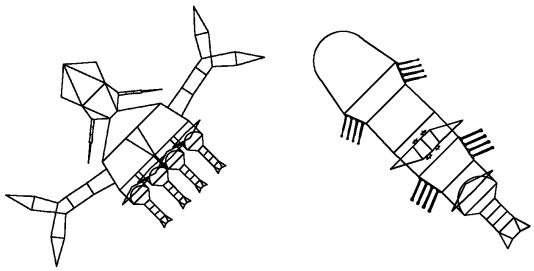


Figure 3.18: Two different generated ships. On the left is a multi-engine "battle cruiser" configuration and on the right is a "gunship" configuration with many cannons.

Chapter 4

Feasibility of Information-Rich Texture

Two studies were carried out in order to determine the feasibility of using texture as a medium to encode information. There are many questions about the suitability of certain texture characters for conveying information and the amount of screen real estate required. The answers to these questions are critical to the development of avatars containing information-rich textures, and these two studies provide a starting point in guiding this development.

The first study focuses on the number of distinct values that can be encoded in each texture characteristic, and the second study focuses on the effect of swatch size on the number of texture levels.

4.1 Study One

The candidate texture characteristics are a set of hand-chosen textures that intuitively seem expressive, but to most effectively use them in avatar design the following question is primary: What is the number of distinct values that can be encoded in each texture characteristic?

A baseline number of values provides a starting point for the feasibility of information-rich textures and is the goal of the first study.

4.1.1 Study Design

The study was designed using texture sets of 250 distinct values that vary a texture characteristic sequentially throughout a range. The values of 0 through 250 are mapped to a linear function of one or more texture generation parameters specifically

chosen to control each texture characteristic. The range of the textures in some cases had an intrinsic maximum and minimum range. For example, thresholding progresses from being completely the primary color to being completely the secondary color and so has a clearly defined range. The parameter range of others were manually chosen by visually inspecting the range in which the texture appeared to be changing in a meaningful way. For example, turbulence could be increased infinitely but the range was manually limited to an area where the changes seemed to be perceptibly meaningful.

There are two tasks required in this study. The first task is identifying when two sets of texture swatches are noticeably different with regards to the texture characteristic. The second task is a test that requires the user to correctly sort each randomly ordered texture swatch into the correct group to ensure the difference was correctly identified. The test would randomize the set of texture swatches the participant was looking at when he or she noticed a difference, and the participant would have to successfully sort all of the texture swatches back onto the correct side. The dependent variable being measured in both parts of the study is the number of correctly identified differences and the number of texture frames required until the difference was successfully identified.

For each texture characteristic parameter from 0 through 250 there were four texture swatches generated. The 4 frames were used to average out the randomness inherent in some texture generation methods. For example, one swatch with a texture value of 100 may be similar to most swatches with value of 80 while a different swatch generated with a value of 100 may more closely resemble most swatches with a value of 120. Since the sorting test task requires the sorting of left and right texture swatches to be completely correct, it will be difficult to succeed with all 8 texture swatches until the random ranges for the left and right texture are no longer overlapping. If there were only 2 texture swatches (1 for each side), the effect of random generation would produce many false positive or negative tests. Also, a greater number of swatches makes it more difficult to correctly sort the swatches through chance.

The texture sets used for this study included a single texture set for each texture characteristic. Each texture swatch was 128 by 128 pixels.

4.1.2 Methods

The first study included 8 participants from the University of Saskatchewan between the ages of 19 and 28 with a mean age of 23.9. This group included 6 males and 2 females. All participants were computer users with reported computer usage ranging from 20 to 45 hours a week.

At the beginning of the session the experimenter would explain the task. At the beginning of each texture set was a training screen that provided the name of the texture characteristic being varied and visual examples of how the texture would be changing (see Figure 4.1). Once the participant was ready, he or she could close the training screen and begin the first task of identifying texture differences with a timer advancing texture values automatically. Once the participant indicated he or she noticed a difference, he or she would complete a brief test requiring them to correctly sort the texture swatches. The participants could take a break during either the training screen or the test tasks as these were not timed. Texture sets were presented in a pre-determined cyclical order and each participant started the cycle at a different texture set.

4.1.3 Apparatus

A Java application was installed on a single desktop PC. Figure 4.2 shows the interface for the timed texture difference task while Figure 4.3 shows the interface for the texture sorting task. Figure 4.1 is an example of a training screen used before each texture set.

4.1.4 Tasks

The study required two main tasks: identifying the perceptible difference and sorting texture swatches.

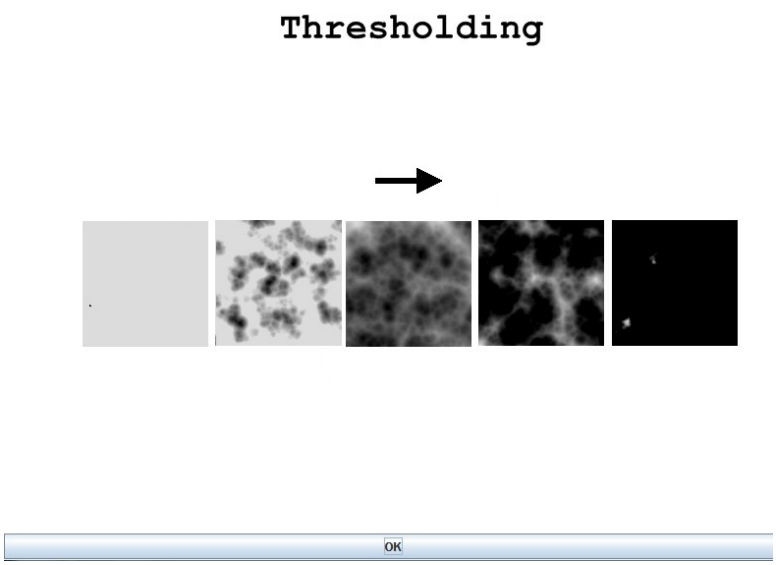


Figure 4.1: An example training screen.

Task 1: Identifying Perceptible Difference

This primary task involved the user observing the two sets of texture swatches. Initially, both texture sets have the lowest parameter value of the texture range (value 0). The parameter value of the rightmost set of texture swatches would be advanced automatically every half second while the leftmost set of texture swatches remains constant. Once the participant can tell the difference between the two texture swatch sets, he or she would press the difference button and then perform the second test task (see Figure 4.2).

When the second task is completed successfully, this first task is repeated again with the two texture swatches being initialized to the previous parameter value of the rightmost set of textures. Successive repetitions of this task advance throughout the range of the parameter values for each texture, and a texture is complete once the rightmost set of texture swatches advances beyond the highest parameter value (250). When all texture are complete, the study ends.

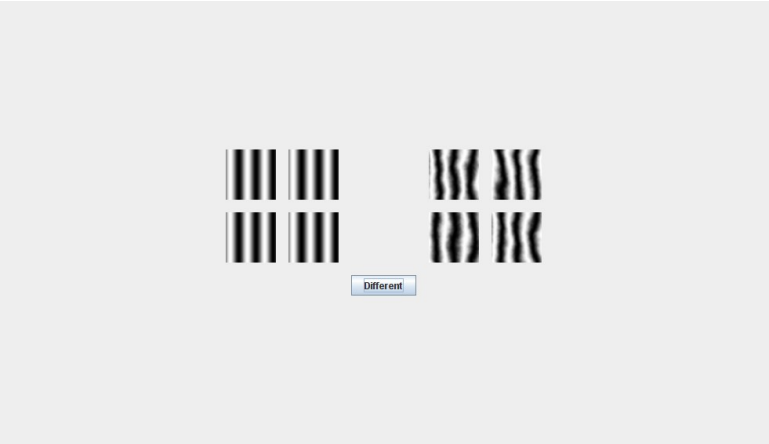


Figure 4.2: The interface for the first task in the study. The participant must press the button once the two texture swatches are noticeably different.

Task 2: Texture Sorting

The secondary task is a check to be sure the participant can accurately distinguish between the two texture sets. In this task all the previous texture swatches from the lower value texture set and the higher value texture set are randomly mixed together (see Figure 4.3). The participant must correctly return all the swatches to their original side. If this step is done correctly, the result is recorded. The result recorded is a pair of values indicating the current texture parameter value of the lower texture set and the change in parameter value the texture required until the difference was detected.

4.1.5 Results

The results of the study are presented in charts showing the change in texture parameter value required to notice a difference throughout the parameter range of the texture. The x-axis is mapped to the total range of the texture sets (0 to 250). The y-axis is mapped to the number of frames the texture series was advanced until a difference was noticed.

The data from all participants was used to calculate a linear regression line which is also plotted on the charts. This linear regression provides a relationship between

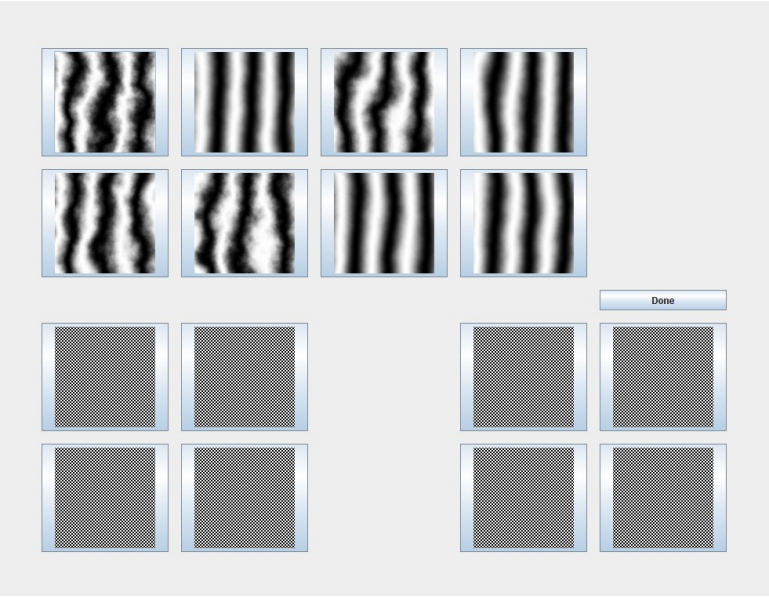


Figure 4.3: The interface for the second task in the study. The participant must correctly sort the shuffled texture swatches at the top into the correct left and right sets.

parameter space and perception space which can be used to estimate the number of distinct values that could be represented with each texture.

A "90% inclusion" line is also plotted on each chart. This line is parallel to the linear regression line, but is drawn high enough that 90% of all participant responses fall below it. The 90% inclusion line provides a safer relationship between parameter and perception space that provides a more conservative estimate of the number of distinct values that could be represented with each texture.

The results for the first study are presented in Figure 4.4 and Figure 4.5 which include the individual responses for each participant as well as the regression and 90% inclusion line.

To provide a general understanding of the usefulness of the textures the analysis examines the total number of levels distinguished by each user and the correlation of all the responses with each other. The average number of levels and the standard deviation is a basic measure to answer the primary question of the study for each texture: how many values can be encoded within each texture? The correlation between all the responses is a metric that reflects how uniform was the performance

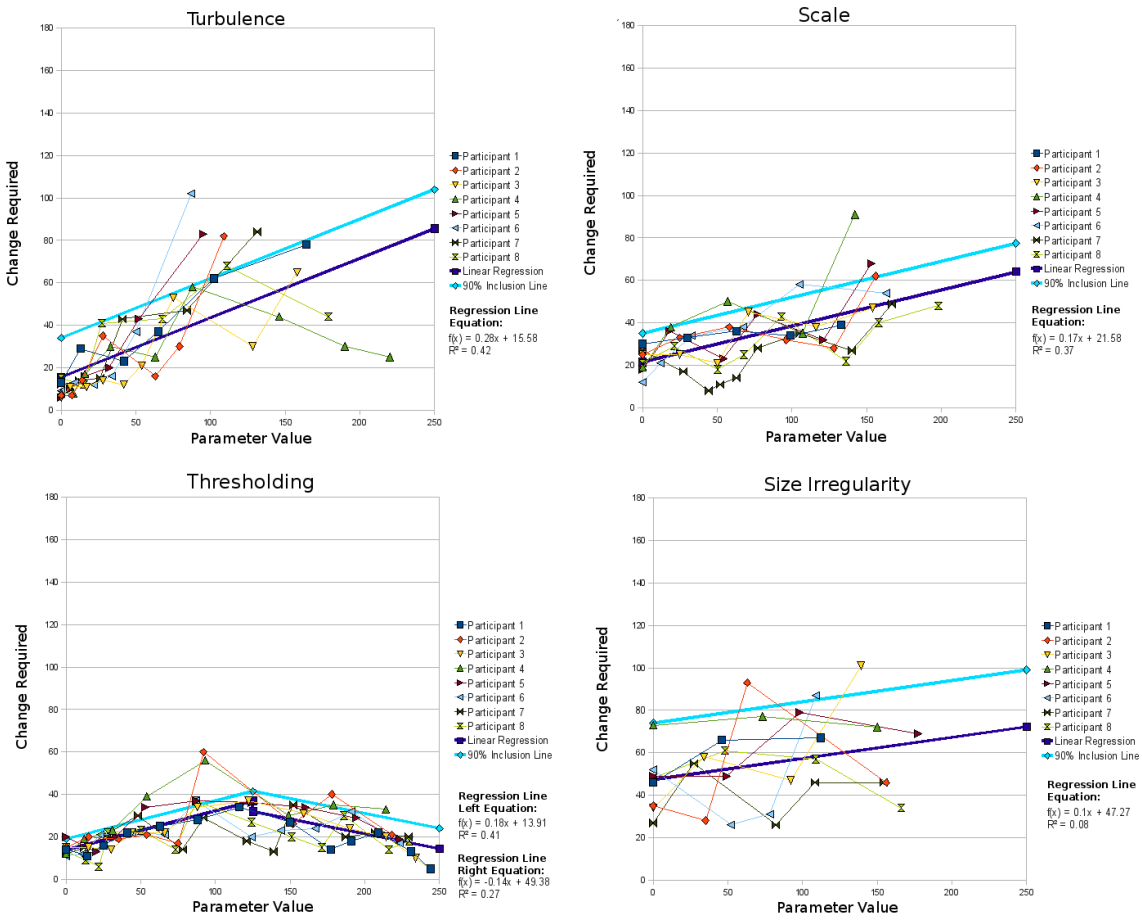


Figure 4.4: Result charts for the first study including Turbulence, Scale, Thresholding, and Size Irregularity.

of the participants. A very low correlation between all responses would indicate that there was a large difference in performance between each participant and that the texture may be more difficult for some people and not as generally useful.

The statistics from the textures used in part one of the study are summarized in Table 4.1.

The average number of levels for each texture is just the average number of successful tests each participant was able to complete for each texture characteristic. It provides an idea of which textures are capable of representing the most distinct values. The textures with the highest capacity were Sparsity and Thresholding with 11.13 and 10.88 respectively. Both of these textures alter the total value of the

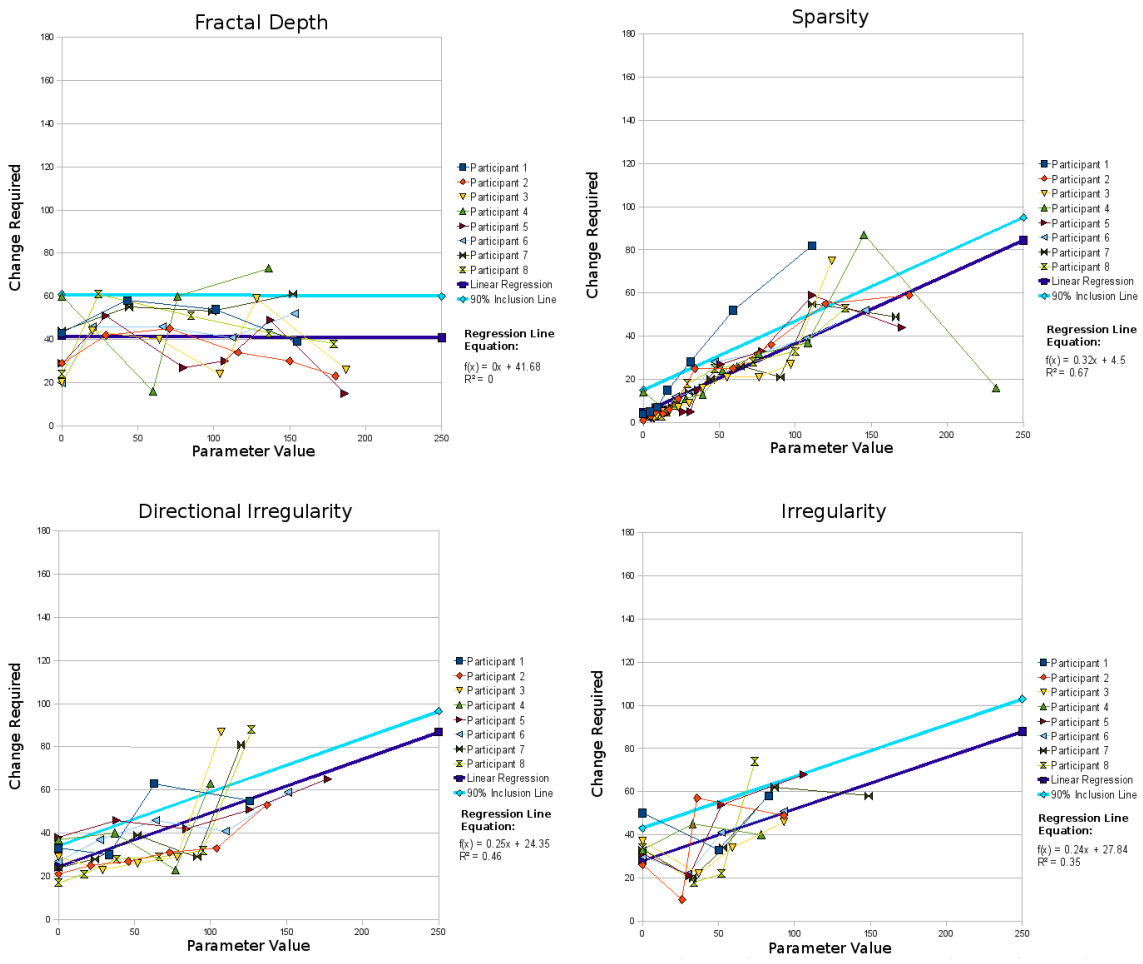


Figure 4.5: Result charts for the first study including Fractal Depth, Sparsity, Directional Irregularity, and Irregularity.

texture (very light to very dark) which seems to indicate that this is one of the most visible aspects of texture. Other textures generally had values about half of those two. Irregularity of textons and texton size were the worst texture characteristics in terms of total levels both with means of 3.88.

The correlation of the various textures are generally in the range of 0.35 and 0.45 although there are a couple of exceptions. The most obvious observation from this analysis is that two textures, Fractal Depth and Size Irregularity, have a very low correlation. This indicates that the required difference between participants for the same value was quite different. This presents a problem when attempting to use this

texture in practice because it is difficult to know what parameter change is sufficient and a worst-case scenario may provide very few usable levels. However, there is indication that training may improve these. A participant in an earlier pilot study (whose data was not included in the main study) was substantially and consistently better than others with the Fractal Depth texture. This seemed to be due to the fact that he was familiar with what fractals were and had an understanding of how they were generated.

Texture	Average Levels	Std. Dev.	Correlation
Turbulence	6.8	1.36	0.42
Scale	6.5	1.69	0.37
Thresholding	10.88	1.96	0.42
Size Irregularity	3.88	0.64	0.08
Fractal Depth	5	0.93	0
Sparsity	11.13	2.03	0.67
Directional Irregularity	5	0.76	0.46
Irregularity	3.88	0.64	0.35

Table 4.1: Summary of statistics for part one of the study.

4.1.6 Study Two

Once a baseline is determined for the information capacity of a given texture, a second question of importance relates to the amount of screen real estate required to maintain it: What effect does texture swatch size have on the number of texture levels?

Screen real estate is a precious resource in any application. The tradeoff between size and information capacity of texture is vital in order to most effectively use display space, and quantifying this relationship is the goal of the second study.

4.1.7 Study Design

This second study was designed in the same way as the first study with a few differences that focused on the size of the texture swatches.

This study used only three texture characteristics: Thresholding, Directional Irregularity, and Turbulence. The texture characteristics Turbulence and Directional Irregularity were chosen because their results from the first part of the study were relatively linearly correlated making changes in performance easier to spot. Also, these two textures have a uniform overall intensity throughout their entire ranges making them more difficult cases and providing a better indication of worst case performance. The swatch sizes used for both these texture characteristics were 64 by 64 pixels, 32 by 32 pixels, and 16 by 16 pixels. The third texture characteristic, Thresholding, was used as well as an example of a texture whose overall intensity varies considerably throughout the range. Only two texture swatch sizes of 32 square and 16 square were used for this texture since it was predicted to be an easier texture (especially given its exceptional performance in part one of the study) and to keep the total time requirements for the second part of the study within bounds.

4.1.8 Methods

The second study included a new and unique group of 8 different participants from the University of Saskatchewan between the ages of 19 and 26 with a mean age of 23.1. This group consisted of 5 males and 3 females. Again all participants were computer users with reported computer usage ranging from 40 to 100 hours a week.

The procedure, apparatus, and tasks for this study were the same as the first study.

4.1.9 Results

The results for this second study are presented the same way as the results for the first study. Every participant response is plotted along with a regression line, and

a 90% inclusion line. The results for the second part of the study are presented in Figure 4.6, Figure 4.7, Figure 4.8.

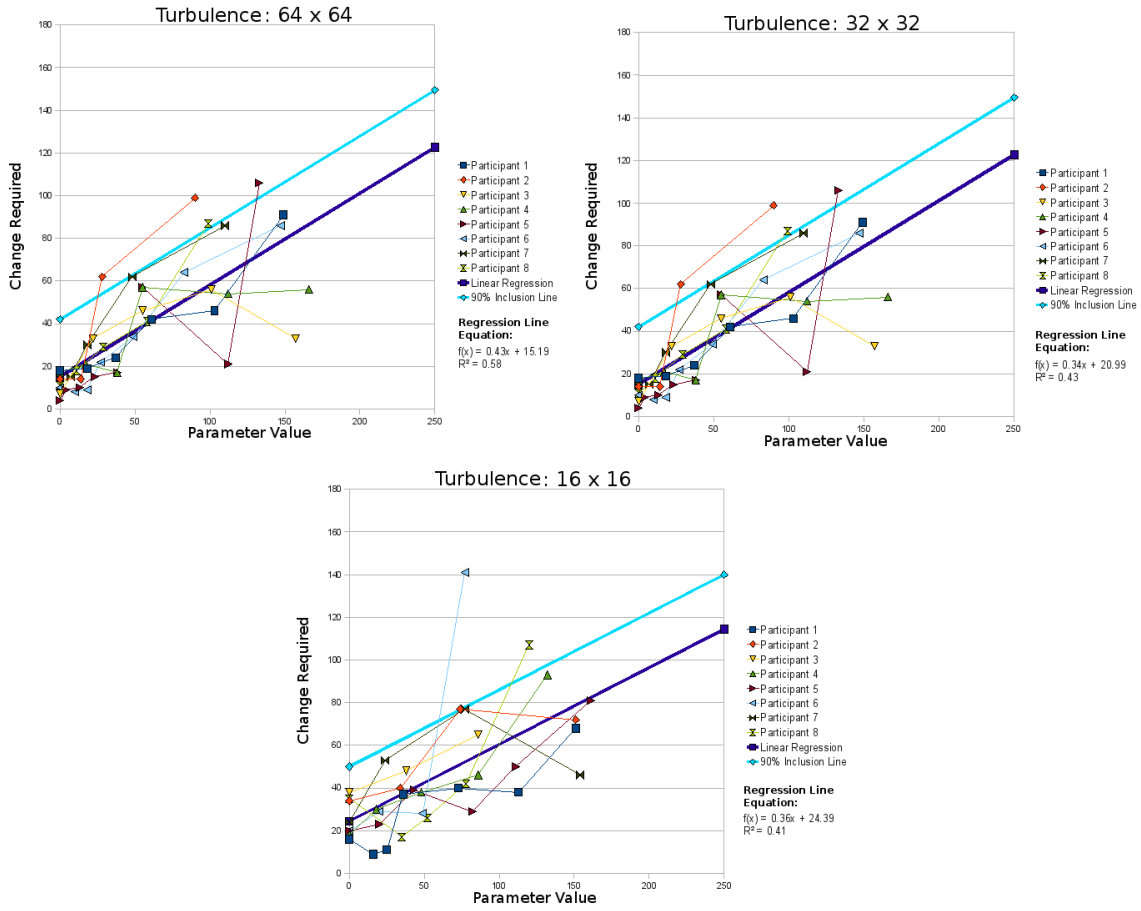


Figure 4.6: Result charts for the second study for Turbulence.

The statistics from the textures used the second study are summarized in Table 4.2. For this table the size of the texture swatch is also given. For each of the texture characteristics used in study two, the results for that texture from study one were also added to the table for reference.

The primary finding of this section is that smaller swatch sizes maintain most of the information capacity of the larger swatches. The smallest Thresholding texture has only 1.56% the area of the largest and yet maintains 74.8% of the texture levels and the correlation is only slightly reduced. The story is similar for Turbulence which reduces the area an equal amount and maintains 69.9% of the texture levels.

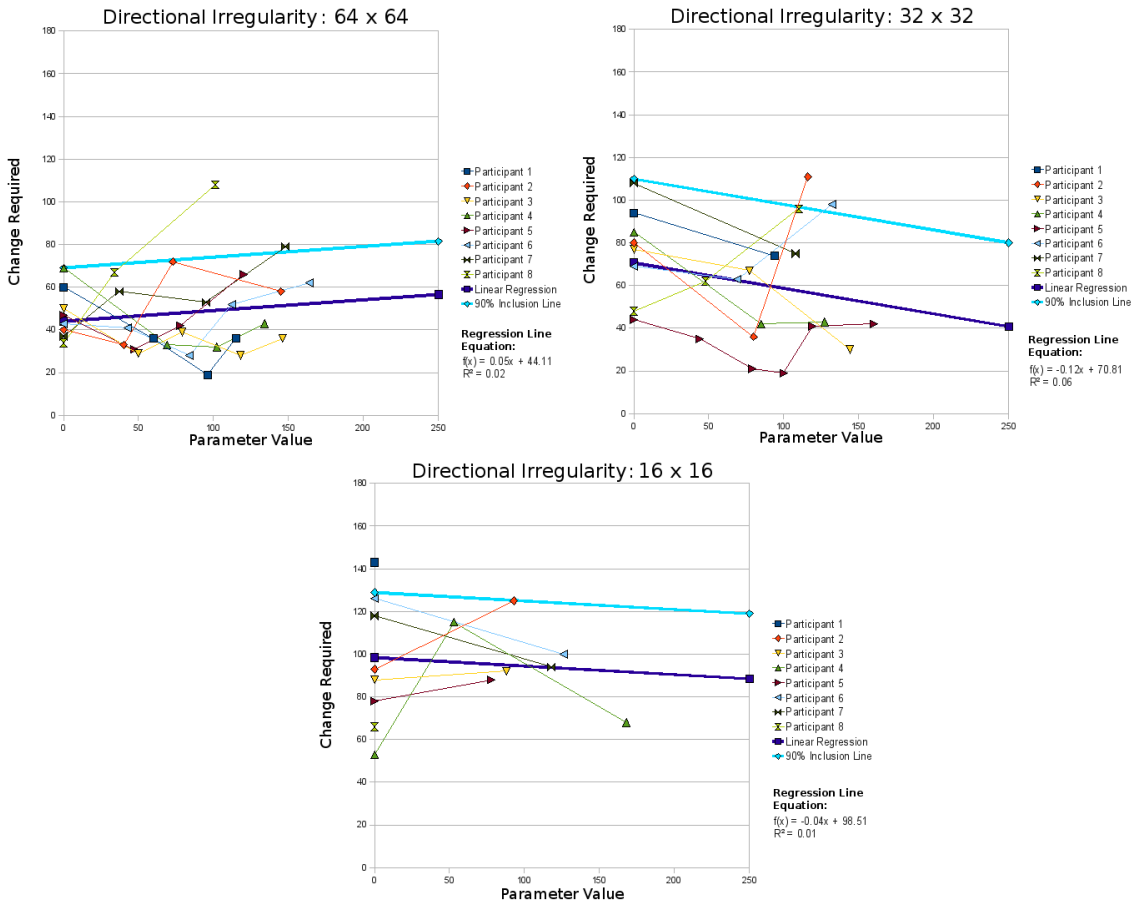


Figure 4.7: Result charts for the second study for Directional Irregularity.

This indicates that textures still maintain their information capacity when reduced to sizes that could easily be included on a large variety of texture embodiments.

It is worth noting that Turbulence is a texture that relies completely on the texture appearance and not the total value of the texture swatch like Thresholding. If you collapsed the Thresholding texture to a single pixel, you would still have a change in range from light grey to complete black. The same is not true of Turbulence–the average value is roughly the same throughout its entire range. Even when Turbulence is very small, it is still possible to distinguish its texture features.

This indicates that texture screen space can be reduced considerably and still maintain the technique's usefulness, and that a textured embodiment would be most

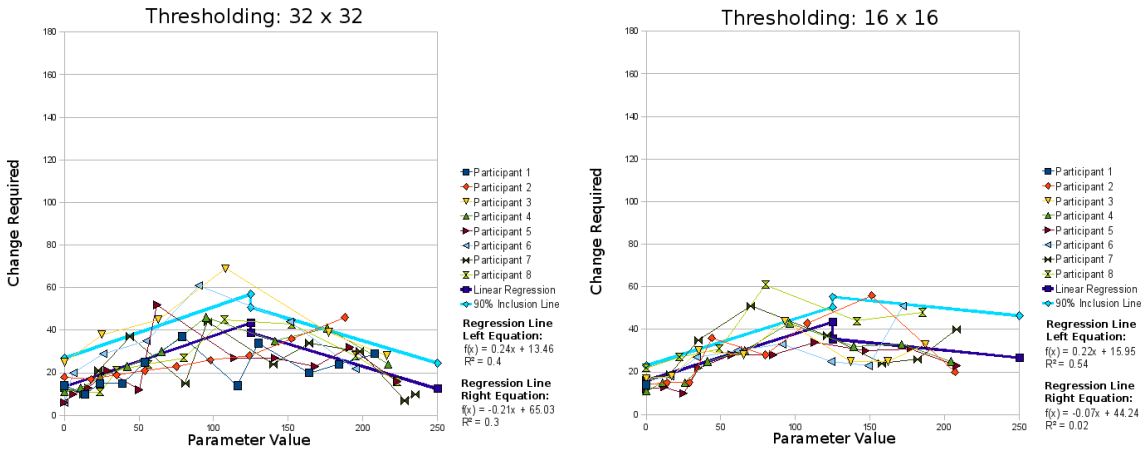


Figure 4.8: Result charts for the second study for Thresholding.

efficiently used if it was subdivided into multiple sections each with a different texture (or with the same texture measuring different variables).

However, there is also a lower limit as indicated by Directional Irregularity. This texture was chosen because it was a more difficult texture with fewer levels, and when the size was decreased it seemed to reach the limit at which its smaller size started to significantly impact its usefulness. The smallest version of Directional Irregularity was very difficult for participants to distinguish and its average did not even exceed two distinct levels (the minimum to consider that a texture has any utility at all).

4.2 Analysis

Further examination of the results was performed by looking at the combined results for each texture with every participant to provide insight into the success or failure of each texture characteristic. The regression line fitted to the data can be used to quantify the relationship between the parameter space and the perception space for an average user throughout each texture characteristic's parameter range. This regression line can be used to calculate the number of distinct levels possible for each texture (see Table 4.3) by successively evaluating its equation starting at a parameter value of 0. The result is added to the parameter value after each evaluation. The

Texture	Average Levels	Std. Dev.	Correlation
Turbulence 128 x 128	6.8	1.36	0.42
Turbulence 64 x 64	5.75	1.28	0.58
Turbulence 32 x 32	5.38	0.92	0.43
Turbulence 16 x 16	4.75	1.28	0.41
Directional Irregularity 128 x 128	5	0.76	0.46
Directional Irregularity 64 x 64	4.13	0.64	0.02
Directional Irregularity 32 x 32	3.13	1.25	0.06
Directional Irregularity 16 x 16	1.88	0.64	0.01
Thresholding 128 x 128	10.88	1.96	0.42
Thresholding 32 x 32	9.25	1.91	0.36
Thresholding 16 x 16	8.14	1.21	0.38

Table 4.2: Summary statistics for study two.

number of times the equation can be evaluated before the parameter value exceeds 250 indicates the total number of levels predicted by the regression line.

Additionally, the 90 percent inclusion line provides a worst case scenario that includes a greater number of potential users. This better takes into account individual performance differences between subjects and textures with responses that varied greatly. It adjusts the results to include more of the poorer performers to provide an estimate of how many distinct levels could be reliably depended on for a given texture.

The results indicate that most textures are capable of handling between 5 and 7 levels of difference. There is a distinct class of texture characteristics including Sparsity and Thresholding that can rely on an average value change of the overall texture (from white to black) which can be used for up to 12 levels.

A second interesting result is the apparent insensitivity of the number of distinct levels to changes in texture scale. Even the difficult texture, Directional Irregularity, which relies entirely on the orientation of small textons (which become nearly indiscernible at the smallest scale used of 16 by 16 pixels) was capable of being used

Texture	Levels	90% Inclusion
Sparsity 128 x 128	11	7
Directional Irregularity 128 x 128	6	5
Directional Irregularity 64 x 64	6	4
Directional Irregularity 32 x 32	5	3
Directional Irregularity 16 x 16	3	2
Fractal Depth 128 x 128	6	5
Turbulence 128 x 128	7	5
Turbulence 64 x 64	6	4
Turbulence 32 x 32	6	4
Turbulence 16 x 16	6	4
Irregularity 128 x 128	6	5
Scale 128 x 128	7	6
Size Irregularity 128 x 128	5	4
Thresholding 128 x 128	12	9
Thresholding 32 x 32	12	8
Thresholding 16 x 16	10	7

Table 4.3: Summary of analysis for both studies.

for 2 distinct levels. This is compared to the maximum of 5 or 6 levels when using larger size swatches.

It is important to note that discrete levels are used to compare textures and provide a metric of suitability for textures to represent information, but the textures are not strictly limited in such a way. One advantage of the texture characteristics is that they are continuous. Some participants were able to detect two or three times the levels of others and to them there may be additional information available even if a texture is only relied upon to represent a small number of values. Participants also had very little training with textures so the number of levels they can perceive will only increase with time as they become more familiar with the representations.

There seemed to be two different classes of texture characteristics that emerged.

There are two that relied mostly upon the changing of the overall texture value from very light to very dark: Thresholding and Sparsity. These two were able to more easily represent additional levels. The other texture characters all relied more upon the textural composition of the image and were generally comparable although more difficult ones did emerge such as Directional Irregularity and Size Irregularity.

4.2.1 Texture Area

The effect of the amount of area used by a texture seemed to have a remarkably small effect on its usefulness. The smallest texture swatch size of 16 x 16 when compared to the largest size of 128 x 128 still maintains an average of two-thirds of the number of identifiable levels despite being only 1.6% of the total area. Thresholding is a texture that affects the overall value of the texture so scale was expected to have less of an effect on it. Thresholding maintained 77.7% of the levels from the smallest to largest swatch size when using the worst-case 90% inclusion. Turbulence was chosen for the second study because it maintained a constant average value over the entire area of the texture, and this requires a user to rely on the change in texture appearance only. This texture had similar performance and maintained 80% of the levels at the smallest texture swatch size. Directional Irregularity does not follow the trend and only maintains about 40% of the number of levels.

Directional Irregularity was chosen for the second study because it was the most difficult of the texture characteristics and the difference between Directional Irregularity and the other two textures can likely be explained by the size of the textons. Directional Irregularity when scaled to 16 x 16 made the individual textons virtually indistinguishable and effectively made the frequency of the total texture swatch too high to be effectively displayed at that resolution. This texture fell below the minimum size for the textons to be individually distinguished which makes their directionality difficult to determine.

This indicates that there is an optimal size for a given texture and going above it provides relatively little benefit. The exact relationship would require more exploration to precisely quantify, but these texture swatches can still be effective while

using relatively small amounts of screen real estate.

4.3 Conclusion

This study took a set of candidate texture characteristics and determined what their potential for conveying information was. Each texture was tested to find how many distinct levels a participant could distinguish, and the textures provided results ranging from 3.8 levels to over 11 levels. This provides a baseline utility for a set of texture characteristics which can be used as a guide to develop future information-rich texture embodiments.

The study also examined the role of total texture swatch size, and found that very small texture swatches retained most of their information capacity. This indicates that textures maintain their usefulness when reduced to sizes practical for use on texture embodiments. Also, the results suggest that, for embodiments with sufficient space, subdividing the space for the use of multiple textures could be beneficial.

Chapter 5

Prototypes

This chapter describes a prototype implementation example within a groupware system that illustrates the application of texture and shape as media for conveying information. Additionally, there are three smaller examples of how these techniques could generalize to different types of information and different systems.

5.1 Prototype Ship in Space

This prototype application was coded in Java and provides only a ship in empty space. It was not coded for optimization or to be actually played, but provides an environment to test out the feasibility and appearance of different effects. The effects are not tied to any attributes of the system but are freely controlled by the user.

5.1.1 Rust

Figure 5.1: A screenshot of the original prototype illustrating the rust effects.

The rust texture is the base texture of the rendered ship and is initially a solid steel color (see Figure 5.1). As the amount of rust increases the thresholding level on the color map for the rust is decreased allowing more of the rust color into the

texture. Ultimately, the color map becomes transparent and as the rust level is increased the ship increasingly disintegrates although it will never fully disappear. The base texture is a cellular texture using 3 octaves of a combination of F_1 and F_4.

5.1.2 Damage

Figure 5.2: A screenshot of the original prototype illustrating the damage effects.

The damage texture is overlayed on the ship's base texture and is manipulated using thresholding (see Figure 5.2). As the amount of damage increases the thresholding level on the color map is decreased allowing the bullet holes to appear and become darker. The base texture is a cellular texture using 5 octaves of the basis cellular functions summed with the following factors: F_1: -1, F_2: 2, F_3: 4, F_4: -5, and F_1 with an exponent of -1.

5.1.3 Normal Map

Figure 5.3: A screenshot of the original prototype illustrating the lighting effects.

The normal map was used with a simple lighting model to provide an illusion of depth to the model (see Figure 5.3). A manually created normal map was developed for the ship and as the parameter was increased a different normal map composed

of Perlin Noise was introduced it to increasingly crinkle the apparent surface of the ship.

5.1.4 Localized Heat

Figure 5.4: A screenshot of the prototype implementation illustrating localized heat, smoke trails, and satellite effects.

Heat textures were added for the primary gun and the engines (see Figure 5.4). The base texture is the same as used for the rust effect except the color map was changed to reflect the appearance of increasingly heated metal. The special aspect to the two texture effects is the local area mask that was used to center the effect at a single point. The color of the texture was modified as the distance from the focal point was increased such that the texture cooled as distance was increased.

5.1.5 Smoke Trails

The smoke trails effect was created by streaming a series of smoke particles out of the back of the ship with a color map that varies from red hot to white smoke to black smoke (see Figure 5.4). A parameter controls the initial color of the smoke and it automatically cools as it exits the ship. Finally, the texture is pasted onto a large background texture which is itself slowly faded out. This results in old smoke slowly fading away in a matter of second or minutes depending on the rate selected. This

effect was fairly resource-intensive in Java so implementation of it for a real system would need some special consideration.

5.1.6 Orbiting Satellites

There were two types of orbiting satellites implemented. The first was a large satellite with a designed shape large enough to provide shape and texture information all by itself (see Figure 5.4). The second type of satellite implemented was a much smaller object only a few pixels wide. This second type of satellite could be used to create a cloud of debris around the ship and could then be used to provide a small amount of texture or color information.

5.1.7 Animated Shield

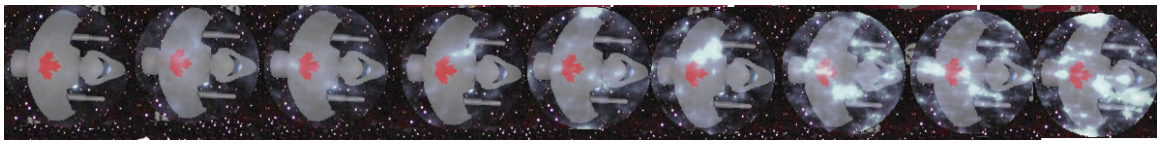


Figure 5.5: Examples of the different levels of the animated shield used in the prototype.

The shield for the ship used an animated cellular texture which is mostly transparent (see Figure 5.5). As the shield amount is reduced the shield is made increasingly transparent. The animation of the shield was implemented to see if additional overlayed textures may be less distracting if they were animated so the underlying textures could be completely seen through different stages of the animation.

5.2 Net Rumble

This application is a functional game built in Microsoft XNA using C#. The game was created by taking a Microsoft demo called "Net Rumble" and extending it to include graphically enhanced spaceship models and additional ship attributes. Game

play involves a type of dogfight in space among several different user-controlled ships at once. This was built as an extension of the original ship prototype into a working, networked game and the addition of a system of parametrically-controlled shape components.

5.2.1 Texture

There are multiple effects rendered on the body of the ship. There are two different types of ships: alien and human. The alien ships have a striped texture while human ships have a metal texture which becomes rusty. Both types of ships also have additional types of texture which include damage, heat, shield, and lighting effects.

Damage

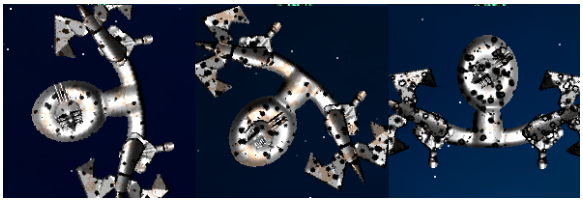


Figure 5.6: Screenshots from the Net Rumble implementation illustrating ship damage.

Damage was visualized using a thresholding type texture and was partitioned onto 5 different areas of the ship: gun, left wing, right wing, engine, and central hull. As damage increases for each ship area, black bullet holes for that area become larger. A secondary effect on the damage texture is damage heat. When damage is inflicted upon a section of the ship, the bullet holes for that section are rendered a hot orange color. As times passes, the orange color fades to black providing an indicator of time since last damage. The hot orange color provides immediate contrast that provides feedback to the player about when attacks penetrate through shields and start damaging the ship (see Figure 5.6).

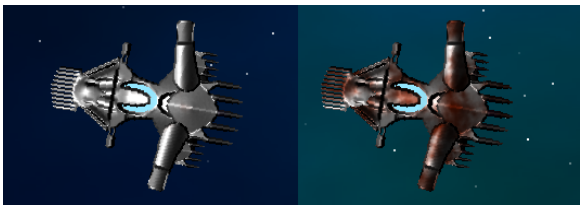


Figure 5.7: Screenshots from the Net Rumble implementation illustrating ship rust.

Rust

The base texture of the human ship is a grey metal with a progressive rust texture which is also a thresholding type texture that increases the amount of reddish-brown rust rendered on the ship. This measures the amount of time that has passed since the ship's last spawn and is potentially useful for spotting skilled or cowardly players that have remained alive for a long time (see Figure 5.7).

Shields

Figure 5.8: Screenshots from the Net Rumble implementation illustrating ship shields.

Shield texture was used to indicate the strength of the shield by the opacity of the shield where a more opaque shield is stronger (see Figure 5.8). Having a mostly opaque overlay interferes with identifying the ship so time was used in two different ways to deal with this issue. The shield texture was animated so that only a portion of the value range of the texture was rendered at any one time. This movement of the texture allows you see the entire ship underneath during different frames of animation. Also, the shield was only rendered for a short time after it was damaged to display the current level, and would quickly fade out within 3 seconds. If a shield slowly fades away, that means that it is still intact to absorb additional damage, but

a shield that instantly disappears after damages is taken or fails to appear at all is depleted.

Alien Skin

Figure 5.9: Screenshots from the Net Rumble implementation illustrating ship striped skin.

There are two distinct types of ships which have different shape and color characteristics. The alien spacecrafts do not use the rust texture, but instead have a lined greenish skin. Instead of rusting like the metallic ships the alien ship grows additional rings on its body. Also, instead of installing shield modules the alien ships skin becomes more complex using the Turbulence texture and its lines become increasingly jagged (see Figure 5.9).

5.2.2 Shape

Figure 5.10: Screenshots from the Net Rumble implementation illustrating different ship types and shape.

A shape grammar was approximated by rendering a ship with multiple different pieces each with pre-rendered growth sequences. The body of a ship determined how

many, where, and at what angle the different components of the ship are connected. Each body section had a turret, wings, and some engines. The body section itself would grow modules and become thicker. Wing segments would become larger and more complex while engines would increase in number and become longer. Turret barrels would increase in number, become longer, and grow larger sensor structures.(see Figure 5.10).

5.3 Examples

The following examples are other situations where visual parametric effects could also be utilized. The first two examples use biometric sensors to map affective data onto embodiments, and the third example uses these techniques in a non-real-time groupware system. These examples are merely potential areas where parametric visual effects may generalize to and provided as inspiration for future work. They have not been evaluated or examined in detail.

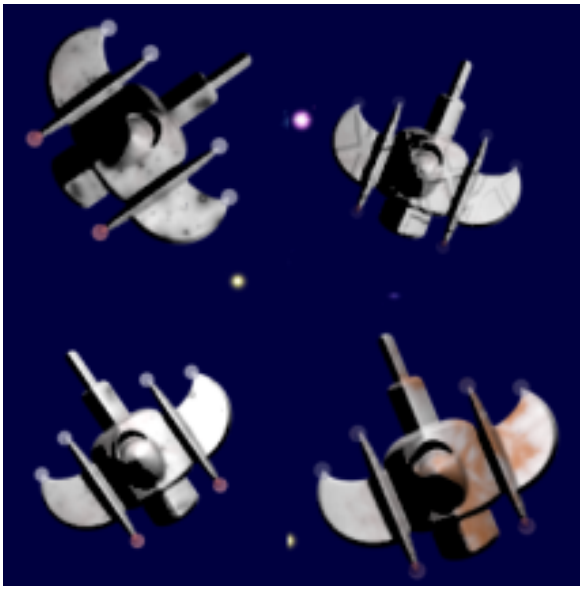


Figure 5.11: Screenshots of the effects used in the affective space combat game including rust, normal map blending, damage, and lights.

This simple ship game is an example of generalizing these rich embodiment techniques to use a different type of information. This used Lightstone (www.wilddivine.com)

sensors which measure heart rate and galvanic skin response. It is an example of potential uses of texture representation of information although in these particular examples the utility of the information was found to be limited and not very relevant to gameplay.

5.3.1 Space Combat

This was a very bare-bones two player space combat game (see Figure 5.11). In this game the heart rate was mapped to the depth of the normal map on the ship, the heart beat itself was mapped to lights on the ship, and the galvanic skin response was mapped to the damage texture of the ship. Sensor feedback was limited to one player.

5.3.2 Poker

Figure 5.12: Screenshots illustrating the techniques used in the affective Poker game. Effects include the pulsating vein, skin splotchiness, and skin color.

This was a simple two player poker game. One character was represented by a human-like avatar with sensor feedback to perhaps provide an advantage in playing poker (see Figure 5.12). The heart beat was mapped to a pulsating vein and the skin

color was mapped to the change in heart rate where blue meant decreasing and red meant increasing. As the galvanic skin response increased the avatar's skin would become blotchier.

5.3.3 Forum

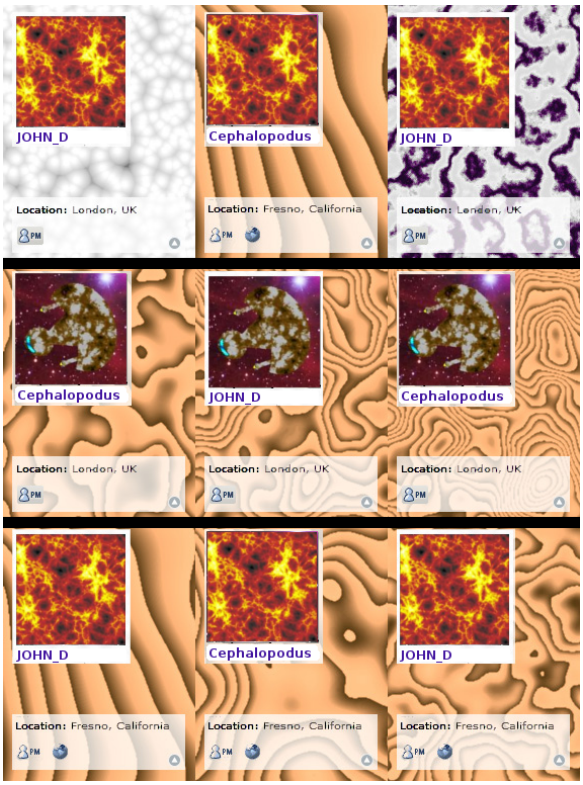


Figure 5.13: A mockup of a possible forum user visualization. The top row is different base texture types: styrofoam, wood, and marble. The middle row shows a change in scale. The bottom show increasing amounts of turbulence.

This example is not an actual implementation, but a mocked up example generalizing parametric visual effects to a different type of system. A forum user representation often includes a photo avatar and some amount of text such as the user's post count, join date, or reputation level if there is some type of feedback system for the type (such as the one used in eBay). The user are often have a rank assigned based on this information. These types of information could be replaced using textures which can be quickly interpreted. Specific details would require an additional more

Figure 5.14: A mockup of a possible forum user visualization. The top row is different amounts of overgrowth, and the bottom row is overlaid with a bullet hole effect.

detailed interface but the specific numbers are usually not important and it is only their comparison with other users that is useful.

There are a number of possible texture augmentations that could be used in a situation (see Figure 5.13). To represent a person's rank in the forum the base texture of the background of the user information panel could be altered. Using a real-world metaphor for hardness the examples I used were styrofoam, wood, and marble which represent increasing ranks within the forum community. The number of posts could be represented by the scale of the texture. Someone with lots of posts will have a large number of styrofoam bubble, wood rings, or marble lines which intuitively represents a large quantity. Also, the complexity of their textured background could be mapped to how long they have been a member. If a user has not logged into the system for a long time (or for threads that have been inactive for a long time) a metaphor of overgrowth could be used and these background textures could develop

an increasing number and thickness of vines (see Figure 5.14). A final example is the bullet hole metaphor which makes a good analog for required moderation for rule violations or bad feedback received from other users. These highly visible bullet holes would instantly identify trolls or scam artists to the other users of the system.

This system was not implemented, but provides another situation and context in which parametric visual effects can be used to enrich and augment a shared virtual space.

Chapter 6

Discussion

In this chapter I will summarize the main results of the research and show how they can be generalized to other areas. The final result will be considered in light of the goal, and the limits of the technique will be discussed.

6.1 Summary of Results

Providing additional information on avatars is useful, but more naturalistic and graphically sophisticated parametric visual effects are required to keep the avatars stylistic consistent. Two parametric visual effects that can help fulfill this requirement are procedural texture generation and generative shape grammars. Both of these techniques can be used to represent information on an avatar.

A set of texture characteristics including Turbulence, Sparsity, Scale, Fractal Depth, Directional Irregularity, Size Irregularity, Irregularity, and Thresholding was examined in more detail. It was found that that the majority of users were able to distinguish between 4 and 7 discrete texture levels. Textures that altered their average value (Sparsity and Thresholding) had a noticeably higher number of texture levels. However, there remain significant individual differences between users where the best user will have 50% to 100% more correctly-identified texture levels. This limits the number of levels a designer can rely upon when creating representations, but more experienced or perceptive users will still be able to make use of their ability to detect finer distinctions. There is also a possibility that more experience with textured representations will lead to improvement among less perceptive users.

The feasibility of these effects has been shown by implementing them in games

and providing static examples. The implementations provide an example and a guide in applying these textures to other systems.

6.2 Generalization

Parametric visual effects can be generalized widely for use with a variety of different tasks, domains, users, and situations because they are general purpose graphical techniques with the ability to represent a large array of objects.

6.2.1 Tasks

These techniques would generalize well to other groupware systems which are not games where it is helpful to be able to easily recognize the identity, status, and actions of other users. For example, systems where people collaborate in real time either to make a document or to simply chat could use these techniques. However, in these situations the importance of highly developed graphics is less important and so these techniques may provide only a supporting or augmenting role for text or icon based visualizations.

Parametric visual effects also have application to asynchronous groupware systems, such as the forum example provided in Section 5.3.3. Without time constraints, the graphical representation is more a matter of system aesthetics and convenience; users may find it quicker to characterize other users by glancing at their representation for information where precision is not required. For example, the exact post count of a user is not important and so it can be shown with a graphical representation that can be quickly compared with others.

Parametric visual effects would also generalize to situations that do not involve user representations at all. They could easily be used to alter the background of the display to represent a system variable, or be applied to certain aspects of the user interface. Applying these techniques in these situations is probably more appropriate to entertainment applications where graphics have a greater value. However, there may be applications where real-world mappings of these effects could be exploited

to increase the understanding of non-experts or young children. For example, the metaphor of rust or foliage overgrowth could be used to indicate that it has been a long time since a virus scan has been run.

6.2.2 Domain

The primary domain used for the examples and implementation of these parametric visual effects was the spaceship, but the techniques are by no means limited to this metaphor. Shape replacement techniques are particularly good at generating plant-like structures, but this technique can be extended to create animals, buildings, or many other shapes. Texture generation techniques are also very general and can be used to create a wide variety of different textures. The spaceship metaphor was used as a convenient example, but these techniques are generalizable to virtually any domain.

6.2.3 Users

The users who participated in the study were all young adults with computer experience, but these techniques should generalize well to a wide variety of users. Parametric visual textures and shapes often take inspiration from naturalistic forms that are familiar to most people. However, there can be large individual differences between users so care must be taken not to exclude less capable users which may belong to any demographic.

However, the studies were both completed by untrained participants who were only briefly introduced to the textures so there is a reasonable expectation that, through training, users would improve. Those users who are more perceptive may learn faster and have greater discrimination. Since texture generation parameters are continuous, more highly skilled users will be able to distinguish more values and the system will be of more value to them.

6.2.4 Situations

In general, parametric visual effect techniques are highly generalizable to other situations. The techniques are flexible and integrate well with the increasing graphical capability of modern computers.

There are many other types of avatars in collaborative applications, and ones with humanoid features are often used. Even with highly realistic human avatars there are skin textures that can be taken advantage of, such as wrinkles, freckles, and veins. It is also common for humanoid avatars to have non-human features which open the domain up to all kinds of unnatural effects, taking inspiration from the animal kingdom, fantasy, and technology, among many others. Although the shape of humanoid avatars may not be commonly altered, there are many types of costumes, armor, tattoos, and other garments that could incorporate additional texture or shape effects.

There is a conflict between customization (i.e. allowing people to decide their avatar's appearance) and using avatar appearance to represent information. However, people do not usually fully utilize all the customization options for avatars, and instead focus on highly visible features that are often customized in real life, such as hair [11]. Reserving these types of avatar features for user customization still leaves a large number of features available for information representation, such as height, weight, and clothing or skin texture.

These effects can also be used on more abstract types of avatars like telepointers or icons in an instant messaging system. These are more abstract situations and so more abstract representations must be used. It is often not difficult to find a reasonable semantic link between an arbitrary texture and a possible variable in the current system. In the forum example provided in Chapter 5 bullet holes were used to represent abstract "damage" done to someone's reputation. A user with a slow network response and high lag could be represented with a highly perturbed and contorted icon indicating they are tangled up and slow. Rust and overgrowth are two metaphors used in examples to represent time lapse or delay, but others could

include dust accumulation or cobwebs.

The downside to using more abstract metaphors for these types of techniques is that they require more learning and provide an opportunity for misunderstanding. The more abstract the representation, the greater this risk is.

Using these very abstract representations increases the number of places this could be used, for example with applications that involve no users at all such as a PC desktop that changes to reflect memory or processor load. These increasingly abstract uses, however, do become less useful and could possibly contribute to confusion and clutter although this may be mitigated somewhat by users interpreting these abstract textures as mere scenery.

6.3 Assessing the Final Result

The final result of avatar creation using parametric visual effects is substantially more realistic and naturalistic than the previous iconic representations (shown in Figure 6.1). There are 23 different variables being represented on the NetRumble spaceship when considering each damage area separately (left wing, right wing, turret, engine, and hull) while the iconic version only included 14. Only 9 of these were on the representation itself while the other 5 text or icon representations could easily be transferred to the textured version if this additional information was required.

The avatar using parametric texture effects is more natural because the textures used are much more realistic. This takes advantage of natural mappings and provides an intuitive understanding of what many of the physical aspects of the ship mean. These more naturalistic representations are less overwhelming for novice users who will see them as merely scenery. This allows for a more gentle learning curve as users become familiar with the information richness of avatars at their own speed.

However, the parametric textured ship is larger than the other ship, and has only limited methods of representing discrete values or text. Additionally, the textured representation adds visual complexity, which may be more intimidating or confusing for a novice user. It may initially be slower to quickly interpret information

from the more complex avatar, but its total information amount is greater once the user becomes familiar with it. There is a trade-off between representing large amounts of information and visual clutter and confusion. The textured ship avatar owes its success partially to the strong domain-specific mapping of the texture to the variables. An equivalent avatar representing completely abstract or arbitrary information would be more difficult to learn.

6.4 Limits

The representation of information is limited by the space available, and space is limited by the visual metaphor used to represent information. The information represented in the textured spaceship worked well because it was strongly correlated to the game metaphor. It would be much more difficult to arbitrarily map these spatially partitioned textures to one of the ship's many attributes. Players would have difficulty remembering if speed was top-left or top-right. This can always be overcome by training, but this partially defeats the purpose of creating naturalistic avatars.

The amount of raw representational power is increased by overlapping or spatially partitioning textures, but the ability for users to tap into this information is dependent on a useful metaphor that allows them to make sense of the visuals. Even a completely abstract representation can be used to represent a few values, and providing a rational connection like the forum example from Chapter 5 can increase this. However, the number of values represented in the Net Rumble spaceship is attributable to strong metaphors and the spatial coherence of the damage location. Using alternate techniques such as icons or text and a few more spatially partitioned objects (such as orbiting satellites) the texture spaceship could likely reach around 30 represented variables that were useful to a player. This is not necessarily a result that will generalize everywhere, and a more abstract program such as a shared word processor would likely be limited to a much lower number.

There is a downfall using complex visual representation because they may actu-

ally provide less information than a traditional method using the same amount of space in that the information represented per pixel may actually be lower. A bar graph is a simple and potentially less ambiguous method of representing information, but a fully featured bar graph is not desirable as an avatar in a realistic virtual world. Information embedded within texture cannot compete with the efficiency of these representations. However, these visual elements have stylistic constraints and simply attempt to maximize the information density of a visualization within these constraints. The goal is to provide information while maintaining the context of the visual appearance of the avatar and without interrupting the flow of the application.

In addition, text and icons can easily be integrated into avatars using parametric visual effects. They increase information content, but also increase the screen clutter and reduce the naturalism of the avatar. These techniques can be very useful when used sparingly for necessary pieces of information. A good use for them is for types of specific information that cannot be conveyed through texture or shape such as names or specific numbers. They are often used in this manner to provide health bars, status icons, and character names.

There are types of information that may not be easy or useful to represent with texture or shape such as a user's current geographical location. In these cases there may be a need for more burdensome representations or a disengagement from the flow of the application to check additional, detailed information. There may also be information which cannot be adequately and completely represented through texture or shape such as a user's numerical experience score or post count in a forum. In these cases a hybrid method could be used where additional, specific information is available upon request but a more general representation is available for quick judgments and broad comparisons.

6.5 Lessons for Designers

The prototypes, design, and results included in this thesis provide many guidelines for designers utilizing information rich avatars. The top lessons for designers from

this thesis are:

1. Designers should make use of parametric visual effects because they can convey information about people in a natural and stylistically consistent way.

2. Several texture examples can carry from three to ten different levels of an information variable.

3. Parametric visual effects can be generalized to many situations and domains even when using abstract mappings.

4. More information can be represented when using less abstract mappings.

5. There is a trade-off between representing more information and introducing visual clutter.

6. Very small areas of texture can represent almost as much information as much larger areas, so it is useful to spatially partition representations.

The primary motivation for using texture for representing information within avatars is maintaining visual coherence with the style of graphics used for highly naturalistic virtual worlds. Avatars that are inconsistent with the naturalistic environment may be distracting, unappealing, and break immersion. The graphical capabilities of modern applications with advanced computer graphics techniques is becoming increasingly important. Additionally, the development and appearance of personal avatars can be a very important method for users to express their personality and individuality. Avatars which are interesting, varied, and appealing are vital to provide the kind of experience that end users will ultimately be interested in.

An advantage to using naturalistic texture generation to represent information is that it can also be perceived by users as simple decoration. A novice user may be overwhelmed by a plethora of numbers, stats, and data but if these are represented through a naturalistic texture, the novice may see them only as visual design elements. The novice can focus on the core elements of the application that they are learning. Once the user becomes more experienced they may learn or discover that certain visual effects or textures are actually representing useful information and they can begin utilizing it.

There seems to be little cost of reducing the size of the texture area compared to

the number of levels that can be effectively represented in it (see Chapter 4) so this seems to imply that it is best to spatially divide the avatar space. However, this is limited by the ability of users to interpret the data. To remain quickly accessible to users the areas must maintain some higher-level semantic meaning such as a wing, arm, or other body part. Arbitrarily placed and very small patches may have value to a computer vision algorithm, but have much less value to a human especially when attempting to quickly perceive a general impression of an avatar.

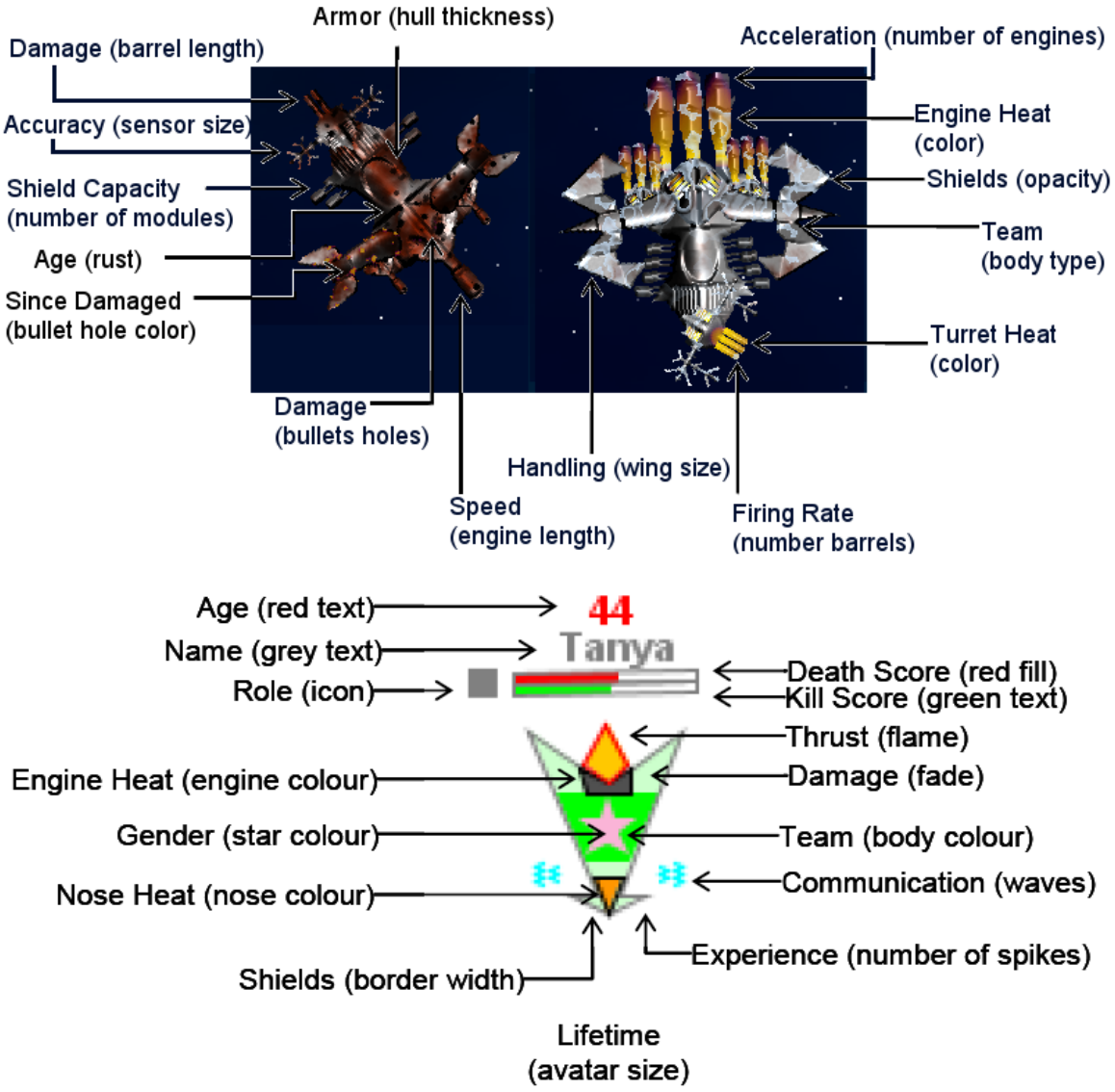


Figure 6.1: Screenshot comparing avatar using parametric visual effects with a similar avatar using iconic representations.

Chapter 7

Conclusion

7.1 Summary

This thesis explored the use of parametric visual effects to represent information on avatars in online environments. Two types of visual effects were investigated: parametric texture generation and shape grammars. Parametric texture generation methods were used to generate a set of texture swatches. The methods used included Perlin noise, Worley noise, and texture splatting. A set of general texture characteristics were chosen that included: Irregularity, Turbulence, Scale, Thresholding, Size Irregularity, Fractal Depth, Sparsity, and Directionality Irregularity. For each texture characteristic an example texture was created and rendered using a large variation of the characteristic.

These texture swatches were used to explore the ability of people to recognize the changes to the underlying parameter as it was reflected in the appearance of the texture. This ability to recognize texture changes could be utilized to represent values in the underlying parameter value. Two studies were done which tested participants' abilities to reliably recognize differences in texture changes. The first study examined each of the eight texture characteristics while the second study looked specifically at only a few characteristics, but included the variation of texture swatch size with the rendered texture scaled appropriately.

These texture characteristics were found to be able to represent many distinct values. The number of values varied from only two values to twelve. These texture characteristics were able to represent almost as much information even at small scales. This indicates that parametric texture generation is a technique that could

be useful for designing information rich avatars for graphically complex visual worlds.

Additionally, shape grammars and L-systems were used to provide examples of potential uses of these methods to also provide a medium in which useful information could be represented in avatars.

Finally, these texture and shape generation methods were implemented in avatars for a number of test systems to explore their feasibility, provide a working example, and a future base upon which to begin further examination of using these techniques in real applications.

7.2 Contributions

The contributions of this thesis include the study of the information capacity of a general set of parametrically generated texture characteristics and the effect of scale on this capacity. The study shows that the concept is feasible, and identified several texture characteristics which have potential to be used for more naturalistic and graphically complex avatar designs.

Additionally, a number of prototype applications were implemented providing some tangible examples of information-rich avatars. These systems provide valuable lessons and a design guide for the development of such avatars for other applications.

7.3 Future Work

This work could be carried forward in several ways such as the introduction of color or animation, the combination of multiple textures, and more detailed studies of these techniques applied and tested in actual systems.

First, the next step would be further studies which test these parametric visual effects with users in real groupware systems. This would require further development of the Net Rumble prototype utilizing the results from the texture studies. The relationship between parameters and perception determined in these studies would inform the use of the texture and parameter ranges used within the system. The

comparison of the predictions made by the study and the real world performance of users would be useful for developing future systems and designing future studies.

Second, one of the most useful and vibrant aspects of texture, color, was purposely omitted from the study of this thesis. Color is a complex subject all to itself and the mere presence of absence of color can be used as an additional variable into any texture. The interplay and combination of colored textures is an area where future work could focus.

Third, all texture characteristics were examined in isolation from each other, but there is potential that multiple characteristics could be utilized within a single texture. Certain textures may destructively interfere with others while some may be completely independent, allowing full parameter ranges for each texture. This combination of multiple textures could potentially provide additional dimensions of information in a single swatch. Fractal Depth combines different amounts of multiple scales of a second texture which itself could be changing based on a second variable. Thresholding could easily be utilized on the Turbulence texture. The interaction of multiple textures evokes many questions on what the maximum potential for a swatch of a given size is and the difficulty for users to easily parse this potentially complex visual information.

Fourth, the textures studied all used stationary, pre-rendered textures, but textures could also be animated. This could provide an additional set of variables such as speed and the type of animation. Animation could also be used to extend the use of multiple texture within a single area. Two textures that may otherwise interfere could be combined by animating one or both of the textures.

References

[1] Christopher Ahlberg. Spotfire: an information exploration environment. *SIGMOD Rec.*, 25(4):25–29, 1996.

[2] Chandrajit Bajaj and Samrat Goswami. Multi-component heart reconstruction from volumetric imaging. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 193–202, New York, NY, USA, 2008. ACM.

[3] Steve Benford, John Bowers, Lennart E. Fahlén, Chris Greenhalgh, and Dave Snowdon. User embodiment in collaborative virtual environments. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 242–249, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[4] Steve Benford, Ian Taylor, David Brailsford, Boriana Koleva, Mike Craven, Mike Fraser, Gail Reynard, and Chris Greenhalgh. Three dimensional visualization of the world wide web. *ACM Comput. Surv.*, page 25, 1999.

[5] Jacques Bertin. *Semiology of graphics*. University of Wisconsin Press, 1984.

[6] Gulen Cagdas. A shape grammar model for designing row-houses. *Design Studies*, 17(1):35 – 51, 1996.

[7] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[8] Mei C. Chuah, Steven F. Roth, Joe Mattis, and John Kolojejchick. SDM: selective dynamic manipulation of visualizations. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 61–70, New York, NY, USA, 1995. ACM.

[9] Lynne Colgan, Robert Spence, and Paul Rankin. The cockpit metaphor. *Behaviour & Information Technology*, 14(4):251–263, 1995.

[10] Stephen Demko, Laurie Hodges, and Bruce Naylor. Construction of fractal objects with iterated function systems. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 271–278, New York, NY, USA, 1985. ACM.

[11] Nicolas Ducheneaut, Ming-Hui Wen, Nicholas Yee, and Greg Wadley. Body and mind: a study of avatar personalization in three virtual worlds. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 1151–1160, New York, NY, USA, 2009. ACM.

[12] David S. Ebert, Kenton F. Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing & Modeling: A Procedural Approach, Third Edition (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, December 2002.

[13] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

[14] S. K. Feiner and Clifford Beshers. Worlds within worlds: metaphors for exploring n-dimensional virtual worlds. In *UIST '90: Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, pages 76–83, New York, NY, USA, 1990. ACM.

[15] Eric Freeman and David Gelernter. Lifestreams: a storage model for personal data. *SIGMOD Rec.*, 25(1):80–86, 1996.

[16] Andr Gagalowicz and Song De Ma. Sequential synthesis of natural textures. *Computer Vision, Graphics, and Image Processing*, 30(3):289 – 315, 1985.

[17] Carl Gutwin and Saul Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Comput. Supported Coop. Work*, 11(3):411–446, 2002.

[18] Thorsten Hermes, Andrea Miene, and O. Moehrke. Automatic texture classification by visual properties. In R. Decker and W. Gaul, editors, *Classification and Information Processing at the Turn of the Millenium*, pages 219–226, March 1–2 2000. Proc. 23rd Annual Conference Gesellschaft für Klassifikation e.V. 1999.

[19] Russel Kirsch and Joan Kirsch. The anatomy of painting style: Description with computer rules. *Leonardo*, 21(4):437–444, 1988.

[20] Niko Kotilainen, Mikko Vapa, Annemari Auvinen, Matthieu Weber, and Jarkko Vuori. P2pstudio: monitoring, controlling and visualization tool for peer-to-peer networks research. In *PM2HW2N '06: Proceedings of the ACM international workshop on Performance monitoring, measurement, and evaluation of heterogeneous wireless and wired networks*, pages 9–12, New York, NY, USA, 2006. ACM.

[21] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Texture sprites: texture elements splatted on surfaces. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 163–170, New York, NY, USA, 2005. ACM.

[22] John P. Lewis, Ruth Rosenholtz, Nickson Fong, and Ulrich Neumann. Visualids: automatic distinctive icons for desktop interfaces. *ACM Trans. Graph.*, 23(3):416–423, 2004.

[23] Aristid Lindenmayer. Mathematical models for cellular interactions in development II. simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*, 18(3):300–315, March 1968.

[24] Alan MacEachren. *How maps work : representation, visualization, and design / Alan M. MacEachren.* Guilford Press, New York :, 1995.

[25] Thomas W. Malone. How do people organize their desks?: Implications for the design of office information systems. *ACM Trans. Inf. Syst.*, 1(1):99–112, 1983.

[26] Jibitesh Mishra and Sarojananda Mishra. *L-System Fractals, Volume 209 (Mathematics in Science and Engineering).* Elsevier Science Inc., New York, NY, USA, 2007.

[27] David A. Nation. Webtoc: a tool to visualize and quantify web sites using a hierarchical table of contents browser. In *CHI '98: CHI 98 conference summary on Human factors in computing systems*, pages 185–186, New York, NY, USA, 1998. ACM.

[28] Carman Neustaedter and Elena Fedorovskaya. Presenting identity in a virtual world through avatar appearances. In *GI '09: Proceedings of Graphics Interface 2009*, pages 183–190, Toronto, Ont., Canada, Canada, 2009. Canadian Information Processing Society.

[29] Ken Perlin. An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, volume 19, pages 287–296, New York, NY, USA, July 1985. ACM Press.

[30] Ken Perlin. Improving noise. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, New York, NY, USA, 2002. ACM.

[31] Ken Perlin and Eric Hoffert. Hypertexture. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 253–262, New York, NY, USA, 1989. ACM.

[32] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman. Lifelines: visualizing personal histories. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 221–ff., New York, NY, USA, 1996. ACM.

[33] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Development models of herbaceous plants for computer imagery purposes. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 141–150, New York, NY, USA, 1988. ACM.

[34] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Softw.*, 11(6):70–77, 1994.

[35] Robert Spence. *Information visualization*. ACM Press, A Division of the Association for Computing Machinery, Inc., Harlow, Essex, England, 2001.

[36] Tadeusz Stach, Carl Gutwin, David Pinelle, and Pourang Irani. Improving recognition and characterization in groupware with rich embodiments. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 11–20, New York, NY, USA, 2007. ACM.

[37] Jason Stewart, Benjamin B. Bederson, and Allison Druin. Single display groupware: a model for co-present collaboration. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 286–293, New York, NY, USA, 1999. ACM.

[38] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. In C. V. Friedman, editor, *Information Processing '71*, pages 1460–1465, Amsterdam, 1972.

[39] John C. Tang and Scott L. Minneman. Videodraw: a video interface for collaborative drawing. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 313–320, New York, NY, USA, 1990. ACM.

[40] M Tapia. A visual implementation of a shape grammar system. *Environment and Planning B: Planning and Design*, 26(1):59–73, January 1999.

[41] Alan M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952.

[42] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 289–298, New York, NY, USA, 1991. ACM.

[43] Steven Worley. A cellular texture basis function. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294, New York, NY, USA, 1996. ACM.

Appendix A

Study Consent Form

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF SASKATCHEWAN
INFORMED CONSENT FORM

Research Project: Perceiving Texture Differences
Investigators: *Shane Dielschneider, Department of Computer Science (966-2327)*
Dr. Carl Gutwin, Department of Computer Science (966-8646)

This consent form, a copy of which has been given to you, is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, please ask. Please take the time to read this form carefully and to understand any accompanying information.

This study is concerned with people's ability to determine differences in various textures.

The Goal of the research is to use texture differences to convey information.

The session will take up to 30 minutes to complete, during which you will have to determine whether two sets of textures are the same or different.

At the end of the session, you will be given more information about the purpose and goals of the study, and there will be time for you to ask questions about the research.

The data collected from this study will be used in articles for publication in journals and conference proceedings.

As one way of thanking you for your time, we will be pleased to make available to you a summary of the results of this study once they have been compiled (they will be made available on the HCI web site, hci.usask.ca). This summary will outline the research and discuss our findings and recommendations.

All of the information we collect from you (data logged by the computer, observations made by the experimenters, and your questionnaire responses) will be stored so that your name is not associated with it (using an arbitrary participant number). Any write-ups of the data will not include any information that can be linked directly to you. The research materials will be stored with complete security throughout the entire investigation. Do you have any questions about this aspect of the study?

You are free to withdraw from the study at any time without penalty and without losing any advertised benefits. Withdrawal from the study will not affect your academic status or your access to services at the university. If you withdraw, your data will be deleted from the study and destroyed. In addition, you are free to not answer specific items or questions on questionnaires.

Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation. If you have further questions concerning matters related to this research, please contact:

Your signature on this form indicates that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a participant. In no way does this waive your legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. If you have further questions about this study or your rights as a participant, please contact:

- Shane Dielschneider — Department of Computer — (306) 966-2327
- Dr. Carl Gutwin, Associate Professor — Dept. Computer Science — (306) 966-8646 — gutwin@cs.usask.ca
- Office of Research Services — University of Saskatchewan — (306) 966-4053

Participant's signature:_____
Date: _____
Investigator's signature: _____
Date: _____

A copy of this consent form has been given to you to keep for your records and reference. This research has the ethical approval of the Office of Research Services at the University of Saskatchewan.

Appendix B

Pre-Study Questionnaire

Perceiving texture differences – Demographics and TLX Questionnaire

Age:_____ Sex: M F

If you are a student, what is your major? _____

Do you have any visual impairments (including color blindness)? Yes No

If so, what? _____

How many hours do you use a computer in a normal week? _____

If you play games, which games or type of games do you play (please list):

_____

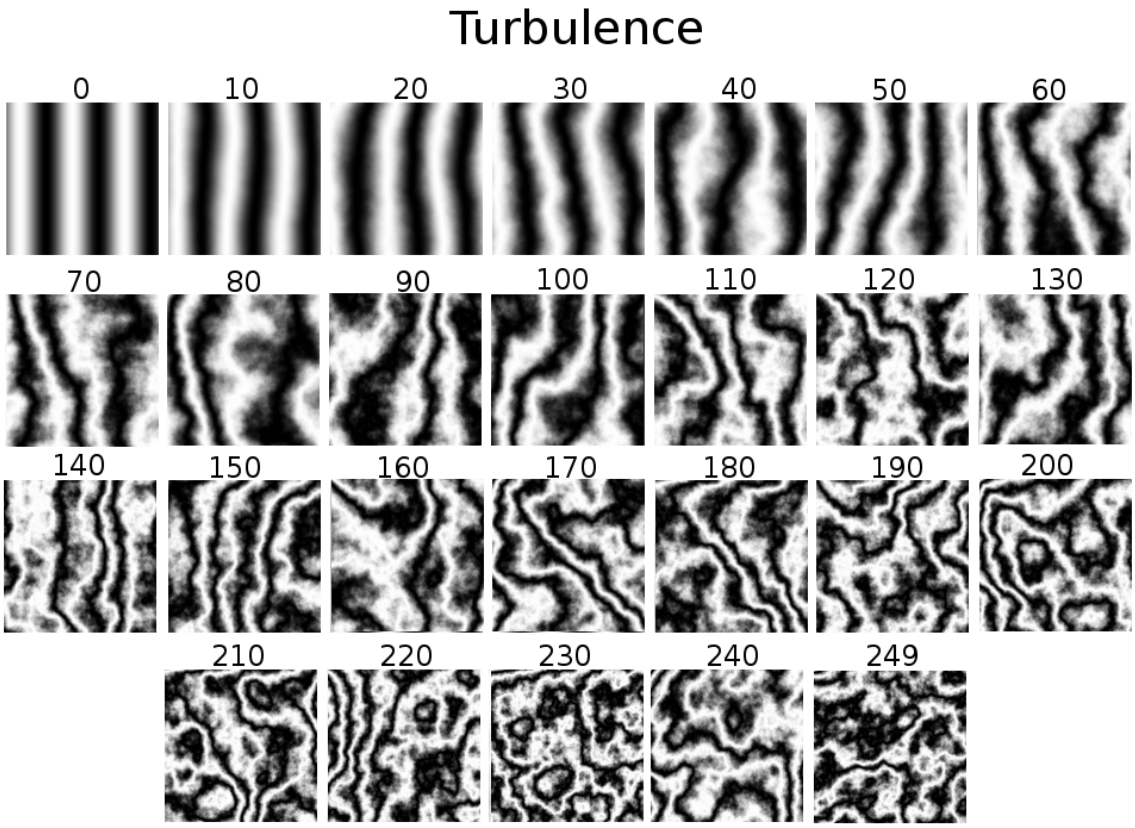
If you have used any of the following devices for computing or gaming please circle them:

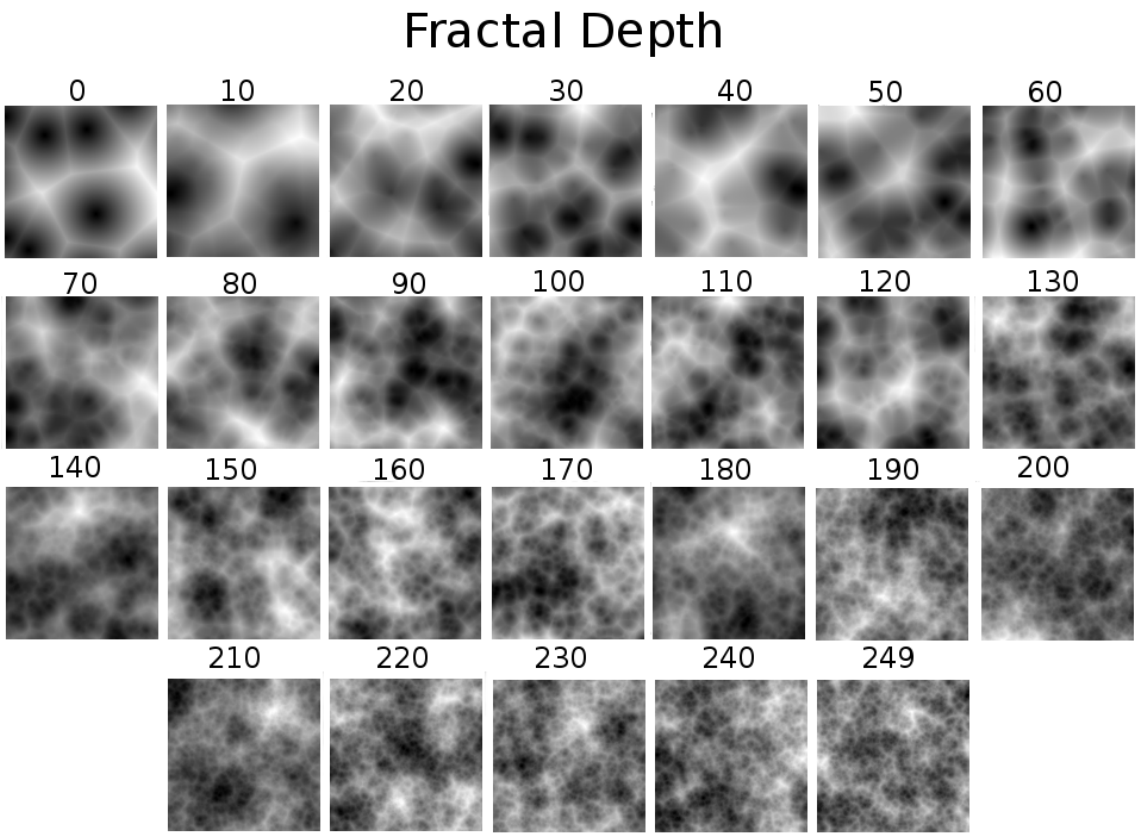
Footpedal Joystick Iso-Joystick (the nub on keyboards or mice) Steering Wheel

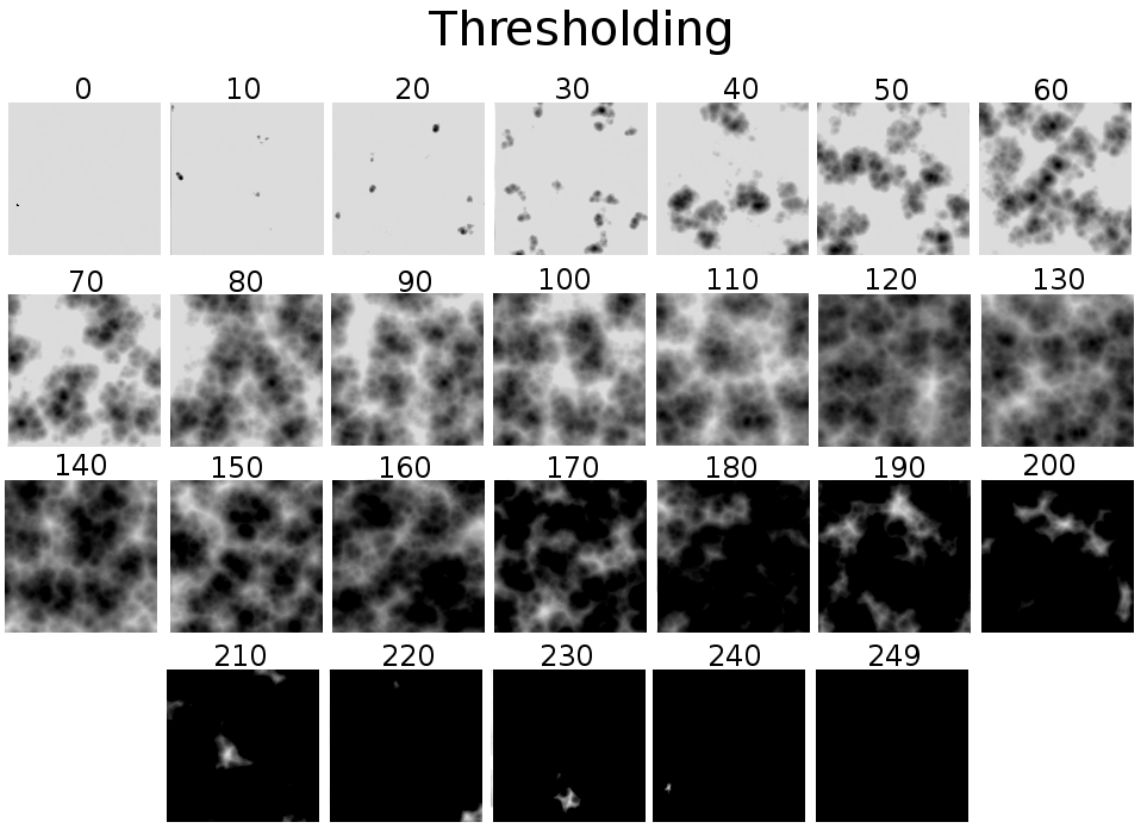
Stylus (PDA/tablet) Xbox 360 controller PS2 controller

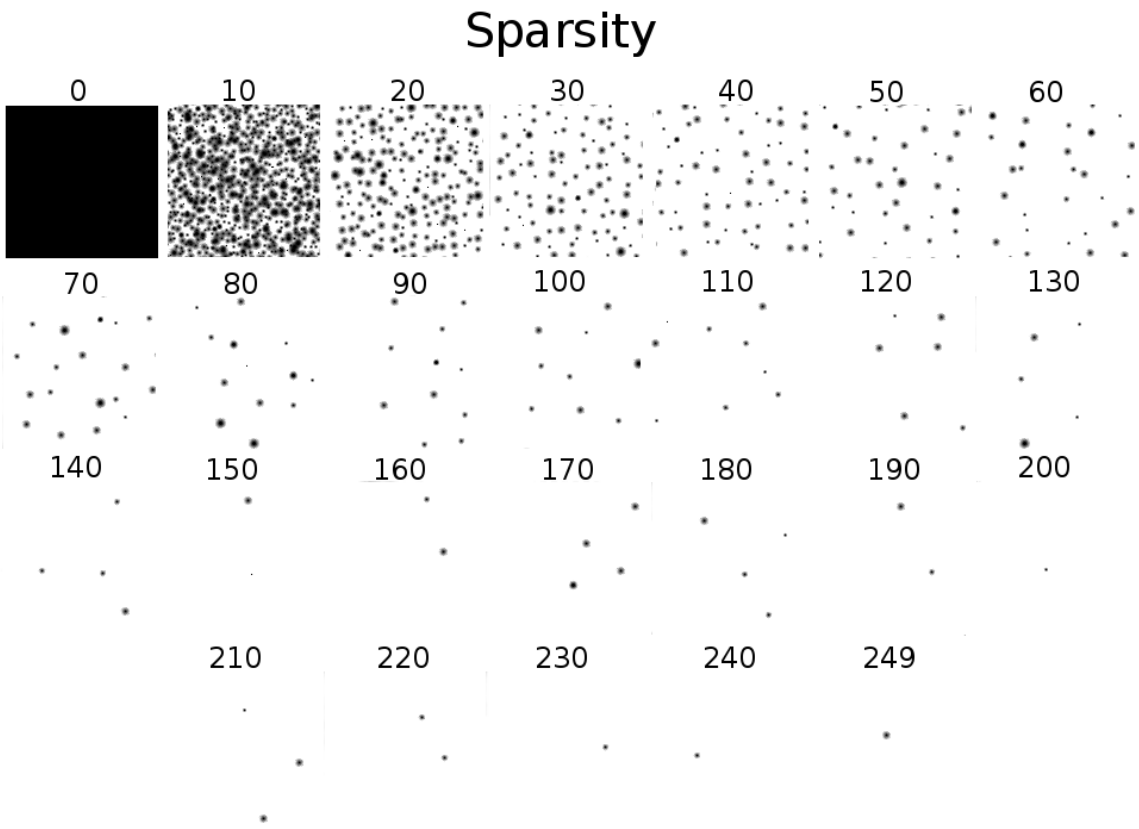
Appendix C

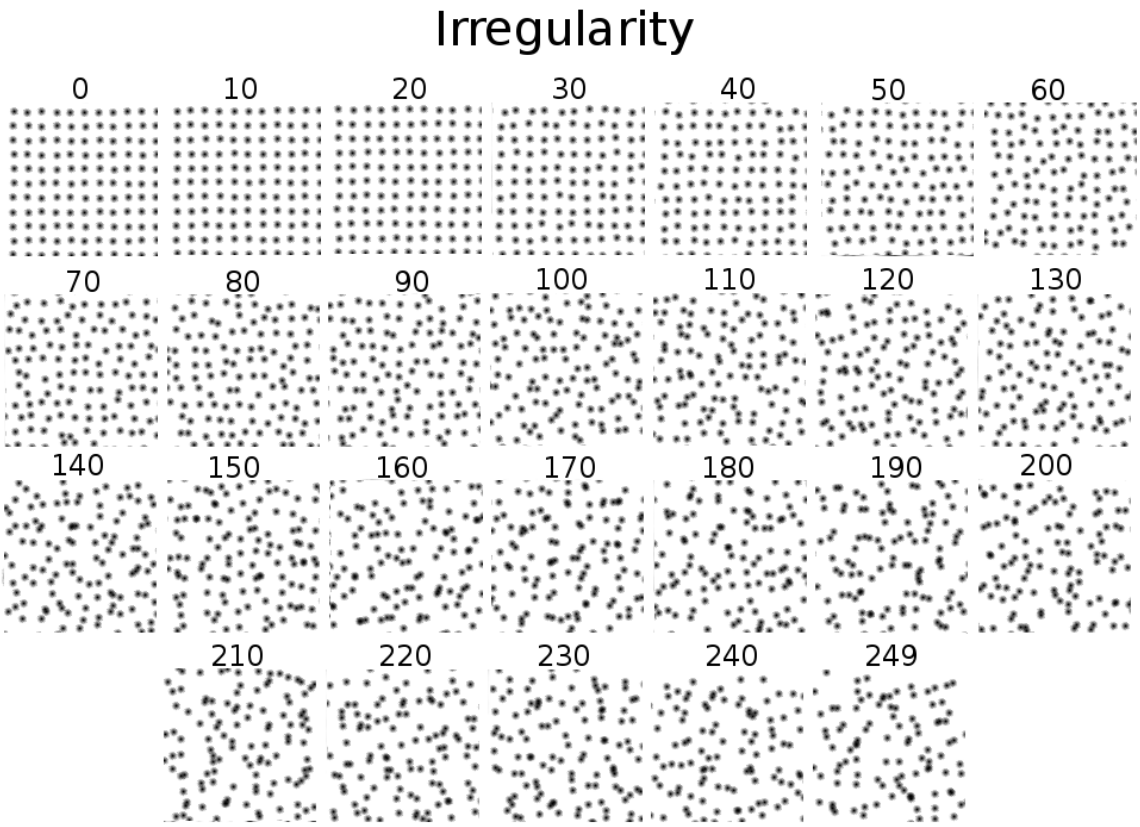
Study Textures

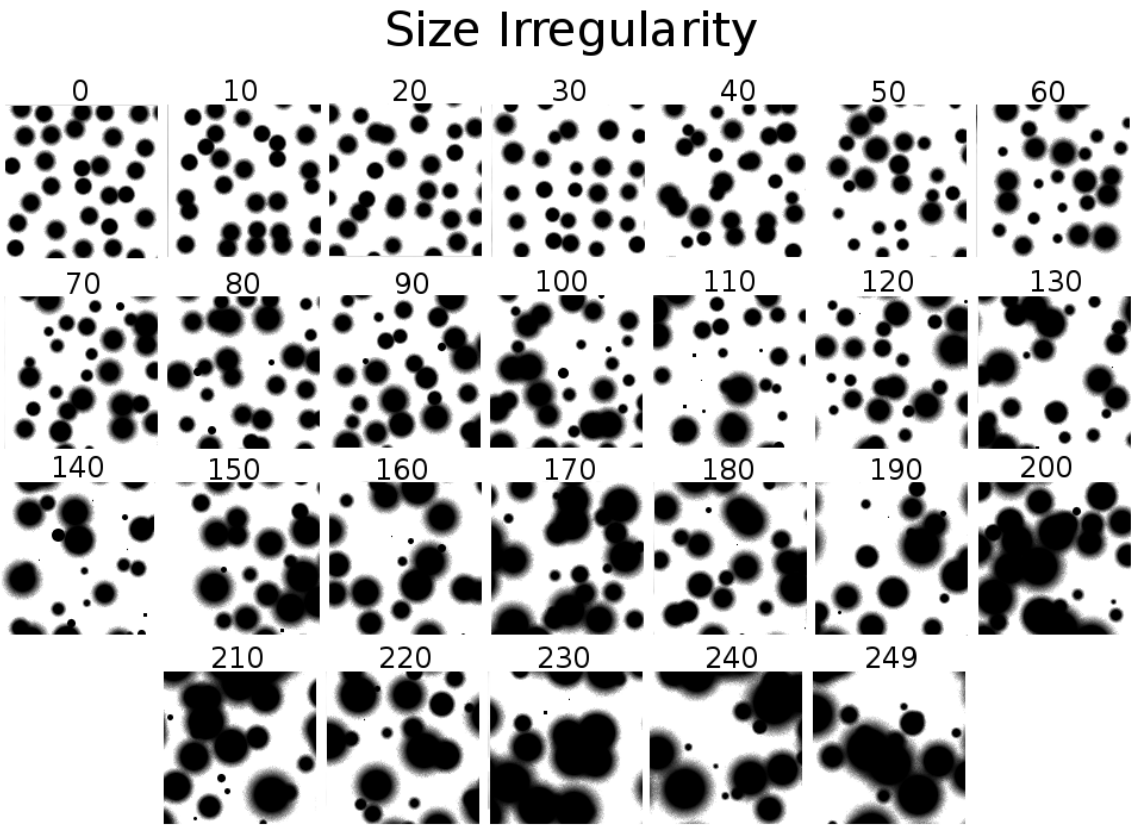


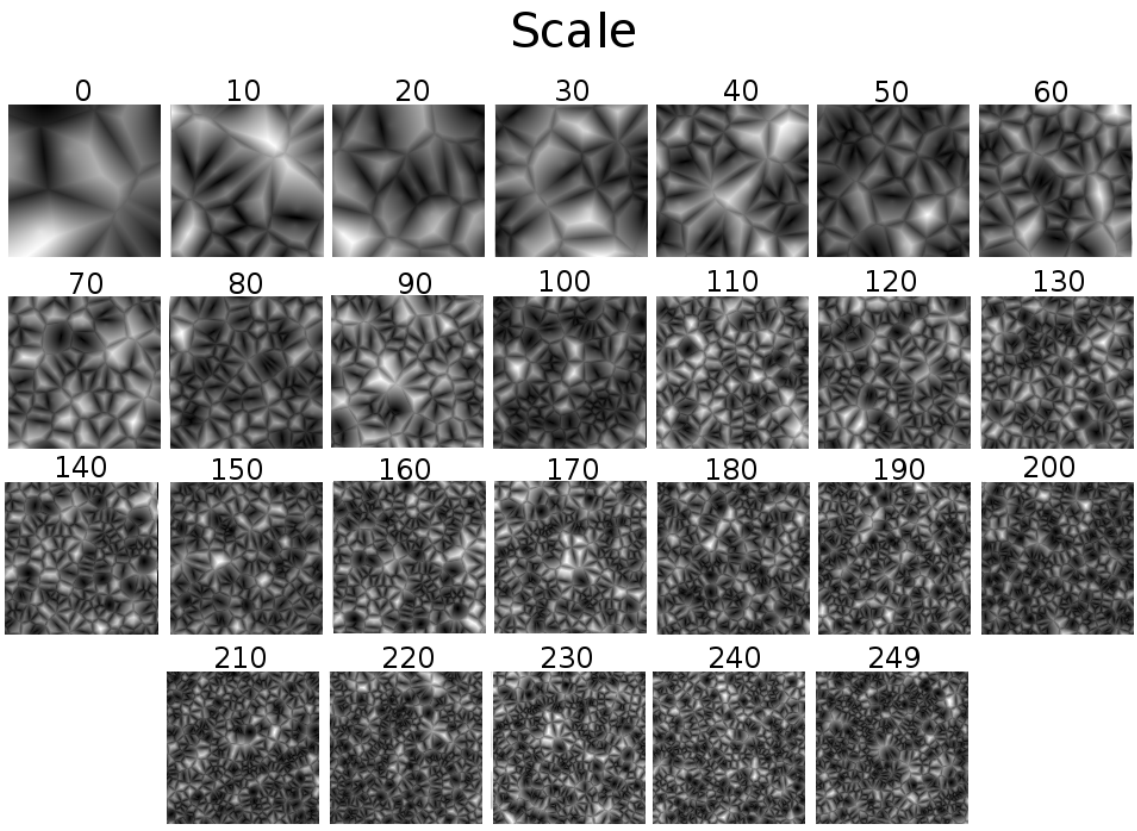

Fractal Depth
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
249

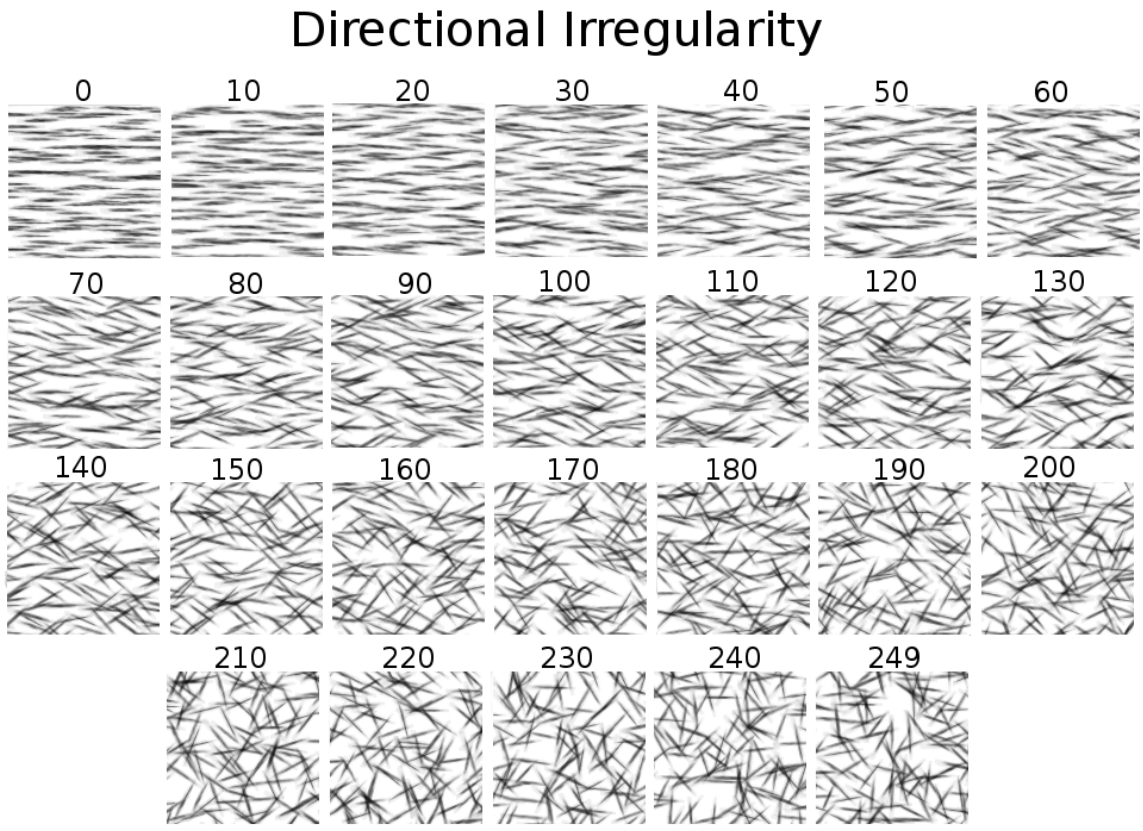

Thresholding
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
249


Sparsity
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
249


Irregularity
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
249


Size Irregularity
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
249


Scale
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
249


Directional Irregularity
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
249